AFHRL-TR-89-43

AIR FORCE HUMAN RESOURCES LABORATORY

TEXTURE DISCRIMINATION RESEARCH USING AN IBM PC

George A. Geri
Christopher D. Voltz

University of Dayton Research Institute
300 College Park Avenue
Dayton, Ohio 45469

OPERATIONS TRAINING DIVISION
Williams Air Force Base, Arizona 85240-6457

March 1990
Final Technical Report for Period October 1987 - July 1989

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This report has been reviewed and is approved for publication.

PAUL M. CHOUDEK, Capt, USAF
Contract Monitor

DEE H. ANDREWS, Technical Director
Operations Training Division

HAROLD G. JENSEN, Colonel, USAF
Commander

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1990	Final - October 1987 - July 1989

4. TITLE AND SUBTITLE

Texture Discrimination Research Using an IBM PC

5. FUNDING NUMBERS

C - F33615-87-C-0012
PE - 61102F, 62205F
PR - 2313, 1123
TA - T3, 03
WU - 12, 83

6. AUTHOR(S)

George A. Geri
Christopher D. Voltz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Dayton Research Institute
300 College Park Avenue
Dayton, Ohio 45469

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Operations Training Division
Air Force Human Resources Laboratory
Williams Air Force Base, Arizona 85240-6457

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFHRL-TR-89-43

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A menu-driven program is described that generates and displays fully textured (i.e., 8-bit gray scale) stimuli which have been used to study human texture perception. The stimuli are displayed on standard video monitors using commercial video-controller cards installed in an IBM PC/AT. The program implements both double random staircase and constant stimuli procedures for obtaining threshold discrimination data and also allows similarity-rating data to be collected. Data in each of these formats can be analyzed and plotted. The program also produces files containing stimulus specifications corresponding to any chosen number of superimposed sinusoids, generates the specified stimulus (with either a rectangular or gaussian window) using the data in those files, and performs a monitor gamma-correction via look-up tables.

Keywords:

14. SUBJECT TERMS

display hardware
laboratory computer software
visual research

15. NUMBER OF PAGES

202

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

SUMMARY

The program described here was designed to present fully textured images as part of a study of human form perception. Use of the program requires only a commercial video-controller board installed in an IBM PC/AT and a standard video monitor. The program generates high-resolution (up to 512 x 512 x 8 bit) images and presents them either singly or in pairs using a procedure which takes into account the subject's previous responses. The program accepts subject responses in the form of numbers entered on a standard computer keyboard. The images generated by the program are composed of sinusoids which are added together to produce texture patterns whose components vary in both orientation and level of detail. The program analyzes and plots the response data and also allows the monitor to be automatically calibrated using light measurement data stored in a look-up table.

PREFACE

This research was performed in support of the Training Technology Planning Objective of the Research and Technology Plan at the Operations Training Division of the Air Force Human Resources Laboratory, Williams Air Force Base, Arizona. The general objective of this training research and development program is to identify and demonstrate cost-effective strategies and new training systems for developing and maintaining combat effectiveness. The purpose of the present experiment was to elucidate the basic mechanisms underlying visually guided behavior in flight simulators.

The authors thank Dr. Yehoshua Zeevi,who provided the techniques and programs used in the texture generation routines. Drs. Don Lyon, Yehoshua Zeevi, and John Uhlarik contributed to the design of the experimental procedures which have been implemented in the programs described in this report. We also thank Dr. Elizabeth Martin for her support and encouragement. This research was supported by the Air Force Office of Scientific Research, Life Sciences, Work Unit 2313-T3-12, Cognitive Aspects of Flight Training, Dr. Elizabeth L. Martin, Principal Investigator; and by Air Force Contract F33615-87-C-0012 (UDRI), Work Unit 1123-03-83, Flying Training Research Support, Capt. Paul M. Choudek, Contract Monitor.

TABLE OF CONTENTS

TEXTURE DISCRIMINATION RESEARCH USING AN IBM PC

I. INTRODUCTION

The program GEXPT2.C, which is described here, has been used to conduct form processing research using an IBM PC/AT in conjunction with a commercial video-controller card (Image Action PCVision or Data Translation Model DT2871) and a standard raster monitor. The program was developed as part of a study of human texture perception performed at the Basic Research Laboratory at the Air Force Human Resources Laboratory, Williams AFB, Arizona. Texture stimuli were generated by adding together various numbers of sinusoidal luminance profiles, each with a specific spatial frequency, orientation, and phase. The resulting stimuli were presented in pairs, with one member of each pair on either side of a fixation point. The observers were asked either to rate the perceived similarity of the stimulus pair by assigning to it a number between 1 and 7 or to depress one of two switches on a response box, attached to the computer's parallel port, depending on which side of the display corresponded to the signal of interest. The program also provides for the presentation of adapting stimuli when the DT2871 board is used. Finally, the program analyzes and plots the data for both experimental paradigms as required.

As the programs described here have been used in their present form as part of an on-going form processing research project, portions of the programs and their documentation are specific to the stimuli and techniques used in that research. The programs are, however, modular in design and may be adapted to other experimental paradigms. Familiarity with the programs' structure and function may be required to make such changes, and in some cases readers may find it more expedient to use individual modules in their own programs.

II. DOCUMENTATION FOR PROGRAM "GEXPT2.C"

GEXPT2.C is a menu-controlled program designed to present visual texture stimuli on a standard raster monitor using a commercial video-controller card installed in an IBM PC/AT. This program (a) initializes the video controller; (b) collects, analyzes, and plots both similarity rating data and threshold discrimination data; (c) creates files which specify the components in a particular texture image; (d) generates texture images using those files; and (e) maintains a look-up table containing monitor calibration data.

The texture images are generated by adding together sinusoidal functions whose spatial frequency, orientation, position, size, and phase can be independently varied. Other images may be used providing that they conform to the format established by Imaging Technology Inc. in their ImageAction software (version 2.0). This format is composed of

three parts: a 64-byte header, a variably sized comment area, and a variably sized data area which contains the image in binary form (see the Image Action User's Guide, Part Number 47-S00003-02, July 1985). As presently configured, the program also requires that the image files be of the form "xxxxxxxy.img" where "xxxxxxx" is the image name entered under option "Add an image name to a set" in the "Modify Set Info" menu, and "y" is either "l" or "r" denoting the left and right components of the image pair.

The following is a general overview of GEXPT2.C which describes the menu system it uses, the associated programs used in its development, and each of the files which make up the program.

Menu System

GEXPT2 is driven by a menu selection system. A list of possible actions is displayed and one letter in each possibility is highlighted (by using a color different from the rest of the text). A prompt is displayed at the end of the list, requesting that a selection key be pressed. When the user presses one of the highlighted characters, that action is taken. All menus throughout the program carry the same basic pattern: A menu title is displayed, a list of options is displayed, and the user selects the action by pressing the appropriate key (case is not significant). To return to a previous menu, the user should press ESC (or x). If an invalid key is pressed, a beep is sounded. Options which lead to another menu have the word "Menu" in the action description. All the menus in the program will now be given, along with a list of possible keypresses for each menu.

The Main menu has the following options: I, d, C, L, M, T, R, x, ESC. It looks like this:

GEXPT 2: Main Menu

Initialize graphics hardware
Collect/Analyze same/different data menu
Collect/Analyze similarity data menu
List Responses
Modify group information
Texture generation menu
Recalibrate monitor menu

Exit program
<ESC> Exit program

Enter Option: __

The Collect/analyze same/different data menu has the following options: A, C, D, G, S, T, x, <ESC>. It looks like this:

GEXPT 2: Same/Different Data Collect/Analyze Menu

Analyze data file
Collect data
Display group sequence
Graph summary file
Set group presentation sequence
Test response box

Exit menu
<ESC> Exit menu

Enter Option: _

The Collect/analyze similarity data menu has the following options: A, C, D, G, S, x, <ESC>. It looks like this:

GEXPT 2: Similarity Data Collect/Analyze Menu

Analyze data file
Collect data
Display group presentation sequence
Graph summary file
Set group presentation sequence

Exit menu
<ESC> Exit menu

Enter Option: _

The Modify group information menu has the following options: A, D, M, x, <ESC>. It looks like this:

GEXPT 2: Modify Group Info

Current Groups: 'group 1', 'group 2', 'group 3', 'group 4'.
Add a group
Delete a group
Modify a group's set entries

Exit menu
<ESC> Exit menu

Enter Option: _

The Modify set info menu (Modify group information submenu) has the following options: A, D, 1, 2, 3, 4, 5, 6, 7, 8, 9, x, ESC. It looks like this:

GEXPT 2: Modify Set Info

Current Sets (page 1 of 1):
SET A (4, 6): g416g (1), g417g (2), g418g (3), g419g (4).

Add an image name to a set
Delete an image name from a set
# of page to display

Exit menu
<ESC> Exit menu

Enter Option: _

The Texture generation/presentation menu has the following options: C, M, p, i, A, F, D, G, E, L, S, x, <ESC>. It looks like this:

GEXPT 2: Texture Generation/Presentation Menu

Create image component info file
Make textured images
Convert image to left/right pair
Convert image file to ASCII file
Convert ASCII file to image file
Flash image pairs
Display one pair of images
Grab (digitize) an image and save it
Extract a subimage from an image
List files
Shell to DOS

Exit menu
<ESC> Exit menu

Enter Option: _

Finally, the Recalibrate monitor menu has the following options: C, L, P, G, R, S, x, <ESC>. It looks like this:

GEXPT 2: Calibration Menu

Change ideal readings
Load calibration file
Print calibration tables
Graph calibration readings
Recalibrate monitor
Save calibration file

Exit menu
<ESC> Exit menu

Enter Option: _

Responding to Prompts

In the GEXPT2 program, data are entered using two types of prompts. The first type is that used in choosing a menu option. The user simply presses one of the highlighted letters and that action is taken. The second type of prompt is that used for entering numbers, strings, etc. When this prompt is issued, anything the user types is displayed using a color different from that in which the prompt was issued. If the backspace key is pressed, the last letter entered will be deleted. If the ESC key is pressed, a backstroke ('\') will be displayed on a new line and everything which was typed up to that point is ignored. Default options are supplied with many of the prompts. To use the default, press enter without any characters preceding it. Also, to lessen the burden of constantly typing in pathnames for filename prompts, the back single quote (`) can be entered to select the default path for the filetype. For summary files and raw data files, the default is .\data. For image files, the default is \scr. For all other filetypes, the default is the directory in which GEXPT2 is located. The program assumes the following directory structure:

```
\
|
|---\GEXPT (contains all of the GEXPT2 files)
|       |
|       |---\GEXPT\DATA (contains the data and summary files)
|       |
|---\SCR (contains the texture images)
```

The program assumes the following file extensions:

.CAL for calibration files
.RAW for raw data files
.SUM for summarized data files (which can be plotted)
.IMG for image files
.ASC for ASCII image files

Also, when displaying images (Texture generation/presentation menu), image filenames may be preceded with two back single quotes ("`") to indicate the name is of the format \scr\g###g?.img where the ### is the number which follows the two back single quotes and the "?" is replaced with "l" (left) for the first image and with "r" (right) for the second image. For example, if "416 and "417 were entered at the respective prompts, it would translate to the image names \scr\g416gl.img and \scr\g417gr.img.

Program Development Environment

The following programs were used in the development of GEXPT2:

TURBO C 2.0, Borland, Inc.). This is the C language compiler which was used to compile the majority of the files. All files ending with .C (C language files) and .H (C header files) were compiled with this program.

TASM 1.0, Borland, Inc.). This is the assembler used to compile all of the .ASM (assembly) files.

MAKE 1.0, Borland, Inc.). This is the utility used to ensure that all the files compiled were up to date. If they were not, it issued the appropriate commands to recompile files and relink them.

TLINK 1.0, Borland, Inc.). This is the linker used to link the object files together.

TD, Borland, Inc.). This is the debugger used to debug the program.

The environment was set up so that C files could be brought into the TURBO C editor and then modified, recompiled, and relinked. The file GEXPT2.PRJ tells TC which files must be recompiled after a change has been made (provided none of the assembly files have been modified). However, as the program became progressively larger, it was no longer possible to run the program from within the environment. So, the file MAKEFILE. was created to tell the MAKE utility how to do the compilation and linking of the GEXPT 2 program. Thus, it is now possible to completely recompile the program simply by typing MAKE.

Program File Structure

The GEXPT 2 program has a large number of files, and keeping track of what each of them does can be difficult. The following is a list of all of the files comprising GEXPT 2, as well as a short explanation of how each file is used.

MAKEFILE. This file contains a list of the files which need to be compiled (or assembled) and a list that specifies the other files upon which they depend. Also, it contains the instructions which must be executed to recompile the program, clean up the directory, save the files to disk, and to retrieve the files from disk. This file is used only by the MAKE utility.

GEXPT2.PRJ This file also contains a list of file dependencies. However, it is used only by the TURBO C integrated project make to determine when files need to be recompiled.

CONSTANT.H This file contains a list of all the general constants used throughout the program. It contains: the constants used for analyzing data (header size, maximum number of groups, etc.), the default file directories, filenames for the general input and output files (such as the name of the file to which the image creation data are written), the image creation/analysis defaults (such as what the default center x value is), menu display constants (such as what color is used to display normal text), response box constants (such as which button indicates a "same" response), system-specific constants (such as what the expansion character is), and general type definitions (such as byte, word, dword, and boolean).

GEXPT2.H This file contains a list of the functions for which GEXPT2.C has the code (and which other routines can call).

GEXPT2.C This file contains the main driver for the GEXPT 2 program. It contains the code which displays the main menu, initializes the graphics hardware, lists the responses from a data file, modifies group information, and modifies set information.

GEXPT2A.H This file contains a list of the functions for which GEXPT2A.C has the code (and which other routines can call).

GEXPT2A.C This file contains the code which displays the texture menu, converts an ASCII image to a binary image, creates the image generation information file, creates left/right pairs from a left version of the image, displays two images, extracts a portion of an image from an image, flashes a sequence of images (listed in another file), grabs an image, converts a binary image to an ASCII image, and exits the program to create an image using one of the external image generation programs. Essentially, this file contains all of the code which is required to execute the actions listed on the texture generation/presentation menu.

GEXPT2B.H This file contains a list of the functions for which GEXPT2B.C has the code (and which other routines can call).

GEXPT2B.C This file contains the code which analyzes the similarity data, collects the similarity data, displays the similarity experiment's group sequence, displays the similarity data collection menu, flashes the screens during similarity data collection, graphs similarity data summary files, randomizes the similarity trials, and sets the similarity group sequences. Essentially, this file contains all of the code which is required to execute the actions listed on the similarity data collect/analyze menu.

GEXPT2C.H This file contains a list of the functions for which GEXPT2C.C has the code (and which other routines can call).

GEXPT2C.C This file contains the code which calculates a look-up table, displays the calibration menu, retrieves the ideal calibration, loads a calibration file, prints the calibration tables, recalibrates the monitor, and saves a calibration file. Essentially, this file contains all of the code which is required to execute the actions on the recalibrate monitor menu.

GEXPT2D.H This file contains a list of the functions for which GEXPT2D.C has the code (and which other routines can call).

GEXPT2D.C This file contains the code which analyzes the same/different data, collects the same/different data, displays the same/different collect/analyze menu, flashes the screens during same/different data collection, graphs same/different data summary files, randomizes the same/different trials, displays the same/different group sequences, and sets the same/different group sequences. Essentially, this file contains all of the code which is required to execute the actions on the same/different collect/analyze data menu.

DT2871.H This file contains the constants and type definitions necessary to interface with the DT2871 graphics card. It also lists the routines which can be called by other routines.

DT2871.C This file contains the code to access a buffer, clear the screen, turn the display on and off, display a screen, initialize the DT2871 board, print the header information from an image, read an image from disk into a screen, read two images (simultaneously) from the disk into a screen, hold a screen on the display for a given number of refreshes, set the write protect for the given bitplanes, stop all board operations, and write the contents of a look-up table. Essentially, this file contains all of the code which is required to program the DT2871 graphics card to provide the standard list of graphical functions.

PCVISION.H This file contains the constants and type definitions necessary to interface with the PCVISION graphics card. It also lists the routines which can be called by other routines.

PCVISION.C This file contains the code which clears the screen, stops digitizing an image, starts digitizing an image, initializes the PCVISION board, prints the image header information, reads an image from the disk into a screen, saves an image from a screen to disk, holds the displayed screen for a given number of refreshes, sets the LUT which will be written, and writes values into a look-up table.

PCWRAP.H This file contains a list of the functions which other routines may call to control the PCVISION board.

PCWRAP.C This file contains the code to redirect the standard list of graphical functions to the specific functions on the PCVISION card. It was created only because the PCVISION card does not support all of the functions the standard list requires. Since the PCVISION module had already been written, it was faster to make this file than to change the original PCVISION.C file.

TOOLBOX.H This file contains a list of the functions which other routines may call and for which TOOLBOX.C contains the code.

TOOLBOX.C This file contains the code to confirm that a message has been seen, get input from the user using a standard interface, print an error message, print an option with the option keyletter highlighted, print a system error message, print a title page with the title centered, swap two elements of any size, and swap two integers. Essentially, this file contains the code for all the functions commonly used by GEXPT 2.

RESPONSE.H This file contains type definitions necessary to interface with the response box. It also lists the functions which other routines may call, and for which RESPONSE.C contains the code.

RESPONSE.C This file contains the code to get a response from the response box and to test the response box. Essentially, this file contains all of the code necessary to interface with the response box.

386MOVE.H This file contains the type definitions required to interface with the protected mode assembly routines. It also lists the routines which other routines may call and for which 386MOVE.ASM contains the code.

386MOVE.ASM This file contains the code to transfer a block of memory from an extended region to a conventional region, to transfer an image from a region of memory to a DT2871 screen, to set an extended or conventional region of memory to a given value, to enter protected mode, to leave protected mode, to turn address line 20 on and off, to empty the 8042's buffer, to handle exceptions while in protected mode, and to print a 32-bit value to the screen while in protected mode.

GEXPT2.BAT This file contains the batch instructions necessary to allow GEXPT 2 to run until it wants to execute one of the protected mode programs, to execute the protected mode program, and to return after the protected mode program has finished.

III. LISTING OF PROGRAMS MAKING UP "GEXPT2"

gexpt2.bat
constant.h
gexpt2.h
gexpt2.c
gexpt2a.h
gexpt2a.c
gexpt2b.h
gexpt2b.c
gexpt2c.h
gexpt2c.c
gexpt2d.h
gexpt2d.c

```
 1: @rem            FILENAME:               gexpt2.bat
 2: @rem            CREATED:                -1/8803.31
 3: @rem            PROGRAMMER:             Christopher Voltz - UDRI
 4: @rem            LAST MODIFIED:  -1/8811.05
 5: @rem
 6: @rem
 7: @rem                        This batch file is setup to run the gabor experiment program
 8: @rem    (GEXPT2).  The program may wish to execute one of two FORTRAN programs
 9: @rem    or to end.  This information is conveyed via the errorlevel variable
10: @rem    and appropriate action is taken.
11:
12:
13: rem @echo off
14: cls
15:
16: :continue
17: rem     *** Run experimental program
18: gexpt2.exe
19:
20: rem     *** Run create programs if specified
21: pause going to check errors
22: if errorlevel = 204 goto rectsum
23: if errorlevel = 203 goto rectind
24: if errorlevel = 202 goto gausssum
25: if errorlevel = 201 goto gaussind
26: if errorlevel = 200 goto end
27:
28: @echo on
29: pause
```

```
   An unspecified error has occurred.
30: @echo off
31: goto end
32:
33: :gaussind
34: rem     *** Create individual images with a Gaussian window
35: run386 txtg.exp
36: if errorlevel = 1 goto error
37: goto continue
38:
39: :gausssum
40: rem     *** Create summed images with a Gaussian window
41: txtsg.exe
42: if errorlevel = 1 goto error
43: goto continue
44:
45: :rectind
46: rem     *** Create individual images with a rectangular window
47: run386 txtr.exp
48: if errorlevel = 1 goto error
49: goto continue
50:
51: :rectsum
52: rem     *** Create summed images with a rectangular window
53: txtsr.exe
54: if errorlevel = 1 goto error

FILE=GEXPT2.BAT    Fri Jun 16 01:38:33 1989    PAGE=1
```

```
55: goto continue
57:
58: :error
59: rem     *** Notify user that an error occured when generating image(s)
60: @echo on
61: pause
```

```
    An error occured while generating the image(s)
62: @echo off
63: goto continue
64:
65: :end
66: echo finished
67: rem     *** User wishes to quit
68: rem cls
```

```
/*****************************************************************************

        FILENAME:               constant.h
        PROGRAMMER:             Christopher Voltz - UDRI
        CREATED:                -1/8805.23
        LAST MODIFIED:          -1/8904.20
        INTERFACE PROTOCOL:     Turbo C 2.0
        REQUIREMENTS:           none.


    This file defines constants, and types for use with the GEXPT program.
Specifically, it is used to provide an interface to the toolbox utility
routines so they do not have to be compiled every time the program does.
Additionally, it allows include modules for the main program to be compiled
separately to speed compilation and reduce compilation memory requirements.

*****************************************************************************/



/************************
 * CONSTANT DEFINITIONS *
 ************************/

    /*** constants required to analyze data ***/
#define HEADER_SIZE     4           /* number of lines before data in .SUM file */
#define MAX_GROUPS      20          /* maximum number of groups                 */
#define NUM_FREQ        6           /* number of frequencies in a file          */
#define TAB_INDENT      5           /* analyze table indention (in spaces)      */

    /*** filename prefixes -- suffixes are assumed ***/
#define DATA_DIR    "\\GEXPT\\DATA\\"   /* directory for output data                */
#define EXE_DIR     "\\GEXPT\\"         /* directory for executable files           */
#define IMAGE_DIR   "\\SCR\\"           /* directory of images                      */

    /*** filenames of general input/output files ***/
#define CREATE_FILE     "CREATE.DAT"    /* component data filename                      */
#define GROUP_DIR       "GROUPS.DIR"    /* name of file containing group directory      */
#define INIT_INFO       "GNEW"          /* screen calibration information               */
#define LOCK_FILE       "LOCK.TMP"      /* filename of lock file                        */
#define TEMP_FILE       "gexpt2.tmp"    /* filename of temporary file                   */
#define TEMP_FILE_2     "gexpt2t.tmp"   /* filename of 2nd temporary file               */

    /*** image creation/analysis defaults ***/
#define CENTER_X_DEFAULT    2.50    /* x center of image (0..SIZE_RANGE)        */
#define CENTER_Y_DEFAULT    2.50    /* y center of image (0..SIZE_RANGE)        */
#define DEFAULT             -1      /* indicates if default is to be used       */
#define EXTRACT_SIZE        32      /* size of matrix to extract from image     */
#define IMAGE_TYPE_DEFAULT  'r'     /* default image type is rectangular        */
#define LUM_DEFAULT         128     /* mean luminance (0..255)                  */
#define MAX_COMPONENTS      450     /* maximum number of components per image   */
#define MAG_DEFAULT         1.0     /* magnitude (0..1)                         */
#define MAX_LUM_DEFAULT     255     /* maximum luminance (0..255)               */
#define MAX_IMAGE_SIZE      256     /* maximum size of image in pixels          */
```

```
 55: #define SIZE_RANGE          5.0     /* maximum coordinate value                */
 57: #define SIZE_DEFAULT        256     /* resolution in pixels (0..255)           */
 58: #define STEP_TYPE_DEFAULT   'l'     /* default step type is logarithmic        */
 59: #define WIDTH_DEFAULT       2.50    /* width of image (0..SIZE_RANGE)          */
 60:
 61:     /*** menu display constants ***/
 62: #define DEFAULT_MENU    2                   /* displays the current default menu                */
 63: #define ENTRY_COLOR     YELLOW              /* user's text is printed in this color             */
 64: #define ENTRY_INDENT    5                   /* number of spaces to indent data entry prompts    */
 65: #define ERROR_COLOR     LIGHTRED + BLINK    /* error messages are printed in this color         */
 66: #define EXIT_MENU       0                   /* used to exit a menu or the program               */
 67: #define MAIN_MENU       0                   /* will cause main menu to be displayed             */
 68: #define MENU_COLOR      WHITE               /* normal text (menus) are printed in this color    */
 69: #define OPTION_COLOR    CYAN                /* highlighted text (options) are in this color     */
 70: #define REDISPLAY       5                   /* used to redisplay a menu                         */
 71: #define TEXTURE_MENU    1                   /* will display texture menu                        */
 72:
 73:     /*** response box constants ***/
 74: #define LEFT_BUTTON         R_BUTTON_3  /* mask for left button                 */
 75: #define RIGHT_BUTTON        R_BUTTON_1  /* mask for right button                */
 76: #define R_BIT_MASK          0xF0        /* mask to clear extra bits             */
 77: #define R_FOUR_BUTTONS      1           /* notify software four buttons in use  */
 78: #define R_BUTTON_1          0x80        /* mask for first button                */
 79: #define R_BUTTON_2          0x40        /* mask for second button               */
 80: #define R_BUTTON_3          0x20        /* mask for third button                */
 81: #define R_BUTTON_4          0x10        /* mask for fourth button               */
 82: #define R_PORT              0x379       /* I/O address of switch port (LPT1:)   */
 83:
 84:     /*** staircase constants ***/
 85: #define DOWN            0           /* symbolic label for down direction        */
 86: #define INVALID         255         /* code to be used for invalid data         */
 87: #define LEFT            0           /* symbolic label for left response         */
 88: #define LEVEL_INC       0.07        /* difference between adjacent levels       */
 89: #define LOWER           1           /* symbolic label for lower staircase       */
 90: #define MAX_GROUP_CHAR  40          /* maximum number of chars for group name   */
 91: #define MAX_LEVEL       4           /* maximum possible level + 1               */
 92: #if 0
 93: #define MAX_NUM_TRIALS  30          /* numbers of reversals to stop after       */
 94: #endif
 95: #define MAX_NUM_CASE    2           /* maximum number of repeats of a staircase */
 96: #define MAX_SETS        30          /* maximum number of sets per group         */
 97: #define MIN_LEVEL       0           /* minimum possible level                   */
 98: #define NUM_NOISE       5           /* number of noise screens (1-9)            */
 99: #define NUM_PAIRS       10          /* number of pairs of images per group      */
100: #define RIGHT           1           /* symbolic label for right response        */
101: #define SETS_PER_PAGE   10          /* number of sets displayed on a page       */
102: #define UP              1           /* symbolic label for up direction          */
103: #define UPPER           0           /* symbolic label for upper staircase       */
104:
105: #define MAX_NUM_TRIALS  700         /* maximum number of trials in one session  */
106:
107:     /*** system specific constants ***/
108: #define ADAPT_BUFFER    2               /* adapting screen is buffer 2          */
109: #define BEEP_DELAY      500             /* duration of error beep in ms         */
```

```
111: #define BEEP_FREQUENCY  100             /* frequency of error beep in HZ           */
112: #define CLEAR_VAL       128             /* intensity to clear screen to            */
113: #define CR              '\r'            /* carriage sequence for end of line       */
114: #define ESC_KEY         27              /* scan code returned by ESC key           */
115: #define EXPAND_CHAR     '`'             /* expands to appropriate default directory */
116: #define FULL            4               /* quad indicating image is full buffer    */
117: #define MASK_BUFFER     3               /* mask screen is buffer 3                 */
118: #define NOISE_BUFFER    0               /* noise screen is buffer 0                */
119: #define NUM_PASSES      5               /* num of passes of random number generator */
120: #define PREVIEW_BUFFER  4               /* preview screen is buffer 4              */
121: #define SIGNAL_BUFFER   1               /* signal screen is buffer 1               */
122:
123:
124:
125: /********************
126:  * TYPE DEFINITIONS *
127:  ********************/
128:
129: #ifndef _BYTE
130: #define _BYTE 1
131: typedef unsigned char   byte;           /* define a (8-bit) byte type   */
132: #endif
133:
134: #ifndef _WORD
135: #define _WORD 1
136: typedef unsigned short int  word;       /* define a (16-bit) word type  */
137: #endif
138:
139: #ifndef _DWORD
140: #define _DWORD 1
141: typedef unsigned long int   doubleword; /* define a (32-bit) word type  */
142: #endif
143:
144: #ifndef _BOOLEAN
145: #define _BOOLEAN 1
146: enum {FALSE, TRUE};                     /* define a boolean type        */
147: typedef byte boolean;
148: #endif
```

```
/*****************************************************************************

        FILENAME:               gexpt2.h
        PROGRAMMER:             Christopher Voltz - UDRI
        CREATED:                -1/8810.18
        LAST MODIFIED:          -1/8901.19
        INTERFACE PROTOCOL:     Turbo C 2.0
        USAGE:                  #include <gexpt2.h>

*****************************************************************************/



/***********************
 * FUNCTION PROTOTYPES *
 ***********************/

int  control_break_handler(void);
byte display_menu(byte *menu);
void exit_program(int exit_code);
void initialize_hardware(void);
void list_responses(void);
void modify_group_info(void);
void modify_set_info(void);
void set_group_sequence(void);


/********************
 * GLOBAL VARIABLES *
 ********************/

extern boolean adapt_flag;
```

```
/*****************************************************************************

        FILENAME:               gexpt2.c
        PROGRAMMER:             Christopher Voltz - UDRI
        CREATED:                -1/8810.18
        LAST MODIFIED:          -1/8904.11
        INTERFACE PROTOCOL:     Turbo C 2.0
        REQUIREMENTS:           IBM PC w/EGA (256K)


    This program was created to display images for basic experimental research
being conducted by Dr. Geri and Dr. Lyon at the Williams AFB 6-1 lab.
    Specifically, Gabor images are created using the CREATE program (written
in Microway NDP FORTRAN 1.0).  The output images are in the IMAGEACTION format,
and as such, they can read by this program.
    When testing is to be done, the operator should first select the Initialize
framegrabber option.  This sets up the board to a known state.  Next, the
operator selects the Enter subject data option.  This allows the operator to
specify which subject is being tested, the session number, and the adapting
frequency.  Finally, the operator may select the Demo option if he intends to
show the subject sample sessions or he may select the Collect data option if
he wishes to actually collect data.  After the Demo or Collect data option has
been selected and the session is finished, the operator may enter any comments
he has about the session followed by a <return>.  Following this, subject data
must be entered again if the operator wishes to Demo or Collect data again.
    Following data collection, the user may select the Analyze data option.
This option allows the analysis of a single file or multiple files.  The
results may be sent to the screen, the printer, a file, or combinations of
these.  The resulting output contains the staircase (if analysis was for a
single file), the mean, deviation, and other relevant statistical information.
    Finally, note the Test response box option which was included to insure
that the response box battery is not dead and that the response box is
responding properly.  The X or <ESC> options allow the user to terminate the
execution of this program.

    The texture manipulation functions include routines to: create images,
given a previously made data file which specifies component information, with
either a Gaussian or rectangular window; display images stored in the
IMAGEACTION format (images are created in this format); and to convert an image
file to an ASCII file.  Note that due to the size of the programs to create the
images, they must be called in a peculiar fashion which removes this program
from memory, runs the program, and returns to this program.  The actual
implementation is to have a batch file (GEXPT.BAT) which calls this program to
start with.  When an image is to be created, this program exits with an error
code other than zero which indicates what type of image is to be created.
Using the IF ERRORLEVEL statement in the batch file, the appropriate program is
executed and then a call to this program is made again.  When the user wishes
to leave the program, the exit code is set to zero which causes the batch file
to exit and return to DOS.  While this method sounds clumsy, it is the only
general way to call the required routines without memory overhead.  Note: a
lock file (LOCK.TMP) is created if a create program was to be executed.  This
allows the program to know if it should return to the texture menu;
additionally, this allows images to be converted to pairs if such was specified.

FILE=gexpt2.c    Fri Jun 16 01:55:28 1989    PAGE=1
```

```
 55:     The calibration menu functions include routines to: change the ideal
 57: calibration reading, load and save calibration information, to print the
 58: calibration information to a printer, to graph calibration information on
 59: the EGA, and to calibrate the display monitor by presenting different
 60: intensity screens and recording the spotmeter reading.  Provisions have been
 61: made to ensure that the user does not leave the menu without saving the
 62: data, unless he specifically requests to do so.
 63:
 64:     Note that this program assumes that an EGA graphic screen dump program
 65: is already active, such as EGAEPSON.  Also, the program DDIR must be in the
 66: current path.  This program lists a directory, with full file information,
 67: in pages.
 68:
 69: ******************************************************************************/
 70:
 71:
 72:
 73:
 74:     /****************
 75:      * HEADER FILES *
 76:      ****************/
 77:
 78:     /*** standard TURBO C header files ***/
 79: #include <conio.h>
 80: #include <ctype.h>
 81: #include <dos.h>
 82: #include <graphics.h>
 83: #include <io.h>
 84: #include <math.h>
 85: #include <process.h>
 86: #include <stdio.h>
 87: #include <stdlib.h>
 88: #include <string.h>
 89:
 90:     /*** program specific header files ***/
 91: #include <constant.h>           /* program constants to define system          */
 92:                                 /* parameters for included files               */
 93: #if DT2871
 94: #    include <dt2871.h>        /* routines to control DT-2871 board           */
 95: #else
 96: #    include <pcwrap.h>        /* wrapper for pcvision routines               */
 97: #endif
 98: #include <response.h>          /* routines for response box control           */
 99: #include <toolbox.h>           /* general utility routines                    */
100: #include <gexpt2.h>            /* header file for this module                 */
101: #include <gexpt2a.h>           /* texture menu include file                   */
102: #include <gexpt2b.h>           /* keyboard data collection menu include file  */
103: #include <gexpt2c.h>           /* recalibration menu include file             */
104: #include <gexpt2d.h>           /* response data collection menu include file  */
105:
106:
107:
108:     /********************
109:      * GLOBAL VARIABLES *

FILE=gexpt2.c    Fri Jun 16 01:55:28 1989    PAGE=2
```

```
110:      ********************/
112:
113: boolean     adapt_flag = FALSE;     /* is adaptive buffer to be used ?  */
114: unsigned    _stklen = 10*1024;      /* create a stack segment of 10K    */
115:
116:
117:
118:
119:     /**********
120:      *  MAIN  *
121:      **********/
122:
123: void main(void)
124:     {
125:     byte    exit_val;                          /* return value from option selected    */
126:     byte    menu;                              /* menu to display                      */
127:
128:
129:     ctrlbrk(control_break_handler);                     /* setup control break handler      */
130:
131:     if (access(LOCK_FILE, 0)==0)                        /* return to texture menu if lock   */
132:         menu = TEXTURE_MENU;                            /* lock file present; otherwise,    */
133:     else                                                /* go to main menu                  */
134:         menu = MAIN_MENU;
135:
136:     while ((exit_val=display_menu(&menu))==REDISPLAY);  /* display the menu until the user  */
137:     exit_program(exit_val==EXIT_MENU ? 200 : exit_val); /* decides to end the program or    */
138:                                                         /* create an image.  then exit with */
139:                                                         /* errorlevel set to what we want   */
140:                                                         /* to do next.                      */
141:
142:     }   /* main */
143:
144:
145:
146:     /************************
147:      * FUNCTION DEFINITIONS *
148:      ************************/
149:
150:
151:
152: /*---------------------------------------------------------------------------*/
153: int control_break_handler(void)
154:
155:     /*      This module is called whenever control break is hit.  It may
156:         terminate the program if the user so wishes.                        */
157:
158:     {
159:
160:     textcolor(MENU_COLOR);
161:     while (kbhit())
162:         getch();
163:     cprintf("\n\r\n%*cDo you wish to terminate program execution? ", ENTRY_INDENT, ' ');
164:     if (toupper(getche())=='Y')
```

```
165:         return(0);
167:     else
168:         return(1);
169:
170:     }
171: /*---------------------------------------------------------------------------*/
172:
173:
174:
175: /*---------------------------------------------------------------------------*/
176: byte display_menu(byte *menu)
177:
178:     /*      This module displays the menu options, gets a keystroke, and
179:         executes the appropriate module.  The return value indicates if the
180:         user wishes to run a create program, redisplay this menu, or exit
181:         the program.  If the input menu level is set to display an alternate
182:         menu, then it is displayed; otherwise, this menu is displayed.      */
183:
184:     {
185:     byte        return_val;                     /* value to return  */
186:
187:
188:         /*** setup the screen ***/
189:     print_title("GEXPT 2: Main Menu\n\n");
190:     print_option("I|Initialize graphics hardware");
191:     print_option("d|Collect/Analyze same/different data menu");
192:     print_option("C|Collect/Analyze similarity data menu");
193:     print_option("L|List responses");
194:     print_option("M|Modify group information");
195:     print_option("T|Texture generation/presentation menu");
196:     print_option("R|Recalibrate monitor menu");
197:     printf("\n");
198:     print_option("x|Exit program");
199:     print_option("<ESC>|<ESC> Exit program");
200:     cprintf("\n\r\n%*cEnter Option: ", ENTRY_INDENT, ' ');
201:
202:         /*** if reentering program after a create, return to ***
203:          *** texture menu                                    ***/
204:     if (*menu==TEXTURE_MENU) {
205:         while ((return_val=display_texture_menu(menu))==REDISPLAY);
206:         return(return_val==EXIT_MENU ? REDISPLAY : return_val);
207:         }
208:
209:         /*** get the user's option ***/
210:     textcolor(ENTRY_COLOR);
211:     switch (toupper(getche())) {
212:         case 'C': while ((return_val=display_keyboard_data_collection_menu())==REDISPLAY);
213:                   return(return_val==EXIT_MENU ? REDISPLAY : return_val);
214:         case 'D': while ((return_val=display_response_data_collection_menu())==REDISPLAY);
215:                   return(return_val==EXIT_MENU ? REDISPLAY : return_val);
216:         case 'I': textcolor(MENU_COLOR);
217:                   cprintf("\n\r\n\n%*cInitializing Graphics board...\n\r\n", ENTRY_INDENT, ' ');
218:                   initialize_hardware();
219:                   break;
```

```
220:         case 'L': list_responses();
222:                   break;
223:         case 'M': modify_group_info();
224:                   break;
225:         case 'R': while ((return_val=display_calibration_menu())==REDISPLAY);
226:                   return(return_val==EXIT_MENU ? REDISPLAY : return_val);
227:         case 'T': while ((return_val=display_texture_menu(menu))==REDISPLAY);
228:                   return(return_val==EXIT_MENU ? REDISPLAY : return_val);
229:         case ESC_KEY:   cprintf("\bX");
230:         case 'X': return(EXIT_MENU);
231:         }   /* switch */
232:
233:         return(REDISPLAY);              /* redisplay this menu */
234:
235:     }   /* display_menu *,
236: /*------------------------- ----------------------------------------------*/
237:
238:
239:
240: /*-------------------------------------------------------------------------*/
241: void exit_program(int exit_code)
242:
243:     /*      This module clears the screen and prints the termination
244:         message.  It is provides a common exit point where system
245:         deinitialization code may be put.  It also removes the temp file.
246:         The given exit_code is returned to the program's caller upon exit.  */
247:
248:     {
249:
250:     unlink(TEMP_FILE);
251:     textmode(LASTMODE);
252:     textcolor(WHITE);
253:     clrscr();
254:     exit(exit_code);
255:
256:     }   /* exit_program */
257: /*-------------------------------------------------------------------------*/
258:
259:
260:
261: /*-------------------------------------------------------------------------*/
262: void initialize_hardware(void)
263:
264:     /*      This module reads in the calibration data and initializes the
265:         green table's LUTs and the registers on the DT-281 board.          */
266:
267:     {
268:     FILE        *file_in;               /* input file for calibration info  */
269:     char        filename[64];           /* filename of calibration file     */
270:     int         index;                  /* general loop variable            */
271:     int         val;                    /* value to put in LUT              */
272:     lut_type    lut;                    /* a LookUp Table (LUT)             */
273:     char        temp[100];              /* temporary string                 */
274:
```

```
275:
277:        /*** initialize board and registers ***/
278:    if (init_board())
279:        return;
280:
281:        /*** get filename of calibration file ***/
282:    sprintf(temp, "Enter filename of calibration file (%s):", INIT_INFO);
283:    strcpy(filename, INIT_INFO);
284:    get_input(temp, "%s", filename);
285:    if (filename[0]==EXPAND_CHAR) {
286:        strcpy(temp, EXE_DIR);
287:        strcat(temp, filename+1);
288:        strcpy(filename, temp);
289:        }
290:    strcat(filename, ".CAL");
291:
292: #if DT2871
293:        /*** program output LUT 0 ***/
294:    if ((file_in=fopen(filename, "rt")) == NULL) {
295:        print_system_error();
296:        return;
297:        }
298:    for (index=0; index<OUT_LUT_SIZE; index++) {
299:        fscanf(file_in, "%*E,%*E,%d", &val);
300:        lut.out_lut[index].red_green = 0x256*val;
301:        lut.out_lut[index].blue = 0;
302:        }
303:    fclose(file_in);
304:    write_lut(OUTPUT_LUT, 0, 0, OUT_LUT_SIZE-1, &lut);
305:
306:        /*** clear all screens we will use ***/
307:    textcolor(MENU_COLOR);
308:    cprintf("%*c5...", ENTRY_INDENT, ' ');
309:    if (!clear_screen(CLEAR_VAL, NOISE_BUFFER)) {
310:        cprintf("4...");
311:        if (!clear_screen(CLEAR_VAL, SIGNAL_BUFFER)) {
312:            cprintf("3...");
313:            if (!clear_screen(CLEAR_VAL, ADAPT_BUFFER)) {
314:                cprintf("2...");
315:                if (!clear_screen(CLEAR_VAL, MASK_BUFFER)) {
316:                    cprintf("1\n\r");
317:                    clear_screen(CLEAR_VAL, PREVIEW_BUFFER);
318:                    }  /* if MASK_BUFFER */
319:                }  /* if ADAPT_BUFFER */
320:            }  /* if SIGNAL_BUFFER */
321:        }  /* if NOISE_BUFFER */
322: #else   PCVISION
323:            /*** program SIGNAL LUT ***/
324:        if ((file_in=fopen(filename, "rt")) == NULL) {
325:            print_system_error();
326:            return;
327:            }
328:        for (index=0; index<LUT_SIZE; index++) {
329:            fscanf(file_in, "%*f,%*f,%d", &val);

FILE=gexpt2.c    Fri Jun 16 01:55:28 1989    PAGE=6
```

```
330:            lut[index] = val;
332:            }
333:        fclose(file_in);
334:        Set_lut(SIGNAL_BUFFER, GREEN_TABLE);
335:        Write_lut(0, LUT_SIZE-1, lut);
336:
337:            /*** program BLANK LUT ***/
338:        val = lut[CLEAR_VAL];
339:        for (index=0; index<LUT_SIZE; lut[index++]=val);
340:        Set_lut(NOISE_BUFFER, GREEN_TABLE);
341:        Write_lut(0, LUT_SIZE-1, lut);
342:
343:            /*** clear screens ***/
344:        clear_screen(CLEAR_VAL, SIGNAL_BUFFER);
345:        clear_screen(CLEAR_VAL, NOISE_BUFFER);
346: #endif  Which graphics board in use?
347:
348:    }   /* initialize_hardware */
349: /*-------------------------------------------------------------------------*/
350:
351:
352:
353: /*-------------------------------------------------------------------------*/
354: void list_responses(void)
355:
356:    /*      This module requests the name of a data file, a frequency and
357:        an orientation and the name of the output file.  It lists the
358:        responses for the given frequency and orientation for each phase.
359:        In addition it prints the mean and standard error for each list
360:        of responses.                                                    */
361:
362:    {
363:    struct node_struct {
364:        int                 info;
365:        struct node_struct  *link;
366:        };
367:    typedef struct node_struct node;
368:
369:    struct head_node_struct {
370:        int                         count;
371:        int                         phase;
372:        int                         sum;
373:        int                         sum_2;
374:        struct head_node_struct *next;
375:        node                        *responses;
376:        };
377:    typedef struct head_node_struct head_node;
378:
379:
380:    FILE        *file_in;
381:    FILE        *file_out;
382:    char        filename[64];
383:    int         freq_comparison;
384:    int         frequency;
```

```
385:    head_node   *head_ptr;
387:    int         index;
388:    int         index_2;
389:    int         orient_comparison;
390:    int         orientation;
391:    int         phase_comparison;
392:    int         response;
393:    head_node   *h_ptr;
394:    head_node   *h_ptr_2;
395:    node        *r_ptr;
396:    node        *r_ptr_2;
397:    char        temp[100];
398:
399:
400:        /* setup screen and get frequency and orientation and filename */
401:    print_title("GEXPT 2: List Responses\n");
402:    get_input("Enter filename of input file:", "%s", filename);
403:    if (filename[0]==EXPAND_CHAR) {
404:        strcpy(temp, DATA_DIR);
405:        strcat(temp, filename+1);
406:        strcpy(filename, temp);
407:        }
408:    strcat(filename, ".RAW");
409:    if ((file_in=fopen(filename, "rt"))==NULL) {
410:        print_error("Could not open specified input file.");
411:        return;
412:        }
413:    get_input("Enter name of output file:", "%s", filename);
414:    if (!stricmp(filename, "p"))
415:        file_out = stdprn;
416:    else
417:        if (!stricmp(filename, "s"))
418:            file_out = stdout;
419:    else {
420:        if (filename[0]==EXPAND_CHAR) {
421:            strcpy(temp, DATA_DIR);
422:            strcat(temp, filename+1);
423:            strcpy(filename, temp);
424:            }
425:        strcat(filename, ".DAT");
426:        if ((file_out=fopen(filename, "wt"))==NULL) {
427:            print_error("Could not open specified output file.");
428:            fclose(file_in);
429:            return;
430:            }
431:        }   /* else */
432:    fprintf(file_out, "Deviant phase file: %s\n\n\n", filename);
433:    get_input("Enter frequency to list:", "%d", &frequency);
434:    get_input("Enter orientation to list:", "%d", &orientation);
435:
436:        /* intialize head node */
437:    head_ptr = NULL;
438:
439:        /* read first line of data */
```

```
440:     while (!feof(file_in)) {
442:     /* Check to make sure this is the same format as in the data collection routine. */
443:         fscanf(file_in, "%*d%*d%*d%*d%d%d%*d%*d%d%*d%*d%*d%*d%d",
444:                &orient_comparison, &freq_comparison, &response,
445:                &phase_comparison);
446:         if (freq_comparison==frequency && orient_comparison==orientation) {
447:             for (h_ptr=head_ptr; h_ptr!=NULL && h_ptr->phase!=phase_comparison;
448:                  h_ptr = h_ptr->next);
449:             if (h_ptr==NULL) {
450:                 if ((h_ptr=malloc(sizeof(head_node)))==NULL) {
451:                     print_error("Insufficient memory.");
452:                     goto error_exit;
453:                     }
454:                 h_ptr->next = head_ptr;
455:                 h_ptr->phase = phase_comparison;
456:                 h_ptr->count = h_ptr->sum = h_ptr->sum_2 = 0;
457:                 h_ptr->responses = NULL;
458:                 head_ptr = h_ptr;
459:                 }
460:             if ((r_ptr=malloc(sizeof(node)))==NULL) {
461:                 print_error("Insufficient memory.");
462:                 goto error_exit;
463:                 }
464:             r_ptr->info = response;
465:             r_ptr->link = h_ptr->responses;
466:             h_ptr->responses = r_ptr;
467:             (h_ptr->count)++;
468:             h_ptr->sum += response;
469:             h_ptr->sum_2 += response*response;
470:             }   /* if */
471:         }   /* while */
472:
473:             /* sort phase list */
474:     for (h_ptr=head_ptr; h_ptr!=NULL; h_ptr=h_ptr->next)
475:         for (h_ptr_2=h_ptr; h_ptr_2!=NULL; h_ptr_2=h_ptr_2->next)
476:             if (h_ptr_2->phase < h_ptr->phase) {
477:                 swap_int(&(h_ptr->count), &(h_ptr_2->count));
478:                 swap_int(&(h_ptr->phase), &(h_ptr_2->phase));
479:                 swap_int(&(h_ptr->sum),   &(h_ptr_2->sum));
480:                 swap_int(&(h_ptr->sum_2), &(h_ptr_2->sum_2));
481:                 r_ptr = h_ptr->responses;
482:                 h_ptr->responses = h_ptr_2->responses;
483:                 h_ptr_2->responses = r_ptr;
484:                 }
485:
486:             /* print results */
487:     if (head_ptr==NULL)
488:         goto error_exit;
489:     fprintf(file_out, "%*c\t\tPhase\n\n", ENTRY_INDENT, ' ');
490:     fprintf(file_out, "%*c\t\t1\t2\t3\t4\t...\n", ENTRY_INDENT, ' ');
491:     fprintf(file_out, "%*c-------------------------------------------------\n",
492:             ENTRY_INDENT, ' ');
493:     for (index=0; index<head_ptr->count; index++) {
494:         fprintf(file_out, "%*c\t", ENTRY_INDENT, ' ');
```

```
495:         for (h_ptr=head_ptr; h_ptr!=NULL; h_ptr=h_ptr->next) {
497:             for (index_2=0, r_ptr=h_ptr->responses; index_2<index; index_2++)
498:                 r_ptr=r_ptr->link;
499:             fprintf(file_out, "\t%1d", r_ptr->info);
500:             }   /* for each phase */
501:         fprintf(file_out, "\n");
502:         }   /* for each line */
503:     fprintf(file_out, "%*c----------------------------------------------------\n",
504:             ENTRY_INDENT, ' ');
505:     fprintf(file_out, "%*cMEAN\t", ENTRY_INDENT, ' ');
506:     for (h_ptr=head_ptr; h_ptr!=NULL; h_ptr=h_ptr->next)
507:         fprintf(file_out, "%2.4f\t", (float)(h_ptr->sum/(float)h_ptr->count));
508:     fprintf(file_out, "\n%*cSTD DEV\t", ENTRY_INDENT, ' ');
509:     for (h_ptr=head_ptr; h_ptr!=NULL; h_ptr=h_ptr->next)
510:         fprintf(file_out, "%2.4f\t", (float)sqrt((h_ptr->sum_2-h_ptr->sum*
511:                 (float)h_ptr->sum/(float)h_ptr->count)/(h_ptr->count-1)));
512:
513:         /* wait for response if necessary */
514:     if (file_out==stdout) {
515:         fprintf(file_out, "\n\n%*cPress any key to continue.\n", ENTRY_INDENT,
516:                 ' ');
517:         getch();
518:         }
519:     else if (file_out==stdprn)
520:         fprintf(file_out, "\f");
521:
522: error_exit:
523:         /* release memory, close files, and return */
524:     for (h_ptr=head_ptr; h_ptr!=NULL; h_ptr=h_ptr_2) {
525:         for (r_ptr=h_ptr->responses; r_ptr!=NULL; r_ptr=r_ptr_2) {
526:             r_ptr_2 = r_ptr->link;
527:             free(r_ptr);
528:             }
529:         h_ptr_2 = h_ptr->next;
530:         free(h_ptr);
531:         }
532:     fclose(file_in);
533:     fclose(file_out);
534:
535:     }   /* list_responses */
536: /*-------------------------------------------------------------------------*/
537:
538:
539:
540:
541: /*-------------------------------------------------------------------------*/
542: void modify_group_info(void)
543:
544:     /*      This module adds, or deletes groups, or modifies the group's
545:         pair entries.  A 'p' suffix on a pair filename indicates that it
546:         consists of both left and right pairs.  A 'b' suffix indicates that
547:         the group consists of pairs of bandwidth stimuli.
548:                                                                         */
549:
```

```
550:    {
552:    struct node {
553:        char        name[MAX_GROUP_CHAR];
554:        char        filename[60];
555:        struct node *next;
556:        };
557:    typedef struct node group_entry;
558:
559:    char        command;
560:    FILE        *file_in;
561:    group_entry *group_ptr;
562:    group_entry *last;
563:    int         num;
564:    int         num_groups;
565:    char        pair;
566:    group_entry *ptr;
567:    char        temp[80];
568:
569:
570:       /* open group directory file and read number of groups */
571:    strcpy (temp, EXE_DIR);
572:    strcat (temp, GROUP_DIR);
573:    if ((file_in=fopen(temp, "rt"))==NULL) {
574:        print_system_error();
575:        return;
576:        }
577:    fscanf(file_in, "%d ", &num_groups);
578:
579:        /* allocate memory for head of list and initialize next pointer */
580:    if ((group_ptr=(group_entry *)malloc(sizeof(group_entry)))==NULL) {
581:        print_error("Insufficient memory.");
582:        fclose(file_in);
583:        return;
584:        }
585:    group_ptr->next = NULL;
586:
587:        /* read in the group names and the associated filename and  *
588:         * add the group entry to the linked list of entries        */
589:    for (last=group_ptr; !feof(file_in); last=ptr) {
590:        if ((ptr=(group_entry *)malloc(sizeof(group_entry)))==NULL) {
591:            print_error("Insufficient memory.");
592:            fclose(file_in);
593:            for (last=ptr=group_ptr; ptr!=NULL; ) {
594:                ptr = last->next;
595:                free(last);
596:                last = ptr;
597:                }  /* for */
598:            return;
599:            }  /* if */
600:        fgets(ptr->name, MAX_GROUP_CHAR+2, file_in);
601:        ptr->name[strlen(ptr->name)-1] = '\0';          /* kill CR at end */
602:        fgets(ptr->filename, 60, file_in);
603:        ptr->filename[strlen(ptr->filename)-1] = '\0';  /* ditto */
604:        ptr->next = last->next;
```

```
605:          last->next = ptr;
607:          fscanf(file_in, "\n");
608:          }  /* for *7
609:      fclose(file_in);
610:
611:          /* print groups, menu, and prompt; execute option */
612:      for (command=' '; command!=ESC_KEY; ) {
613:          print_title("GEXPT 2: Modify Group Info\n\n\n");
614:          cprintf("%*cCurrent Groups: ", ENTRY_INDENT, ' ');
615:          if (group_ptr->next==NULL)
616:              cprintf(" none.");
617:          else {
618:              for (ptr=group_ptr->next, num=80; ptr!=NULL; ptr = ptr->next)
619:                  if (num+strlen(ptr->name)+4>=80)
620:                      num = cprintf("\n\r%*c'%s', ", 2*ENTRY_INDENT, ' ', ptr->name)-1;
621:                  else
622:                      num += cprintf("'%.*s', ", MAX_GROUP_CHAR, ptr->name);
623:              cprintf("\b\b.");
624:              }
625:          printf("\n\n");
626:          print_option("A|Add a group");
627:          print_option("D|Delete a group");
628:          print_option("M|Modify a group's set entries");
629:          putch('\n');
630:          print_option("x|Exit menu");
631:          print_option("<ESC>|<ESC> Exit menu");
632:          cprintf("\n\r\n%*cEnter Option: ", ENTRY_INDENT, ' ');
633:          textcolor(ENTRY_COLOR);
634:          switch(command = toupper(getch())) {
635:              case 'A': cprintf("A\n\r\r\r");
636:                        if ((ptr=(group_entry *)malloc(sizeof(group_entry)))== NULL) {
637:                          print_error("Insufficient memory.");
638:                          for (last=ptr=group_ptr; ptr!=NULL; ) {
639:                              ptr = last->next
640:                              free(last);
641:                              last = ptr;
642:                              }  /* for */
643:                          return;
644:                          }  /* if */
645:                        get_input("Enter name of group to add: ", "", ptr->name);
646:                        get_input("Does the group consist of pairs (but not bandwidth stimuli)? ", "%c", &pair);
647:                        if (toupper(pair)=='Y')
648:                          sprintf(ptr->filename, "%-dP.GRP", ++num_groups);
649:                        else {
650:                          get_input("Does the group consist of bandwidth stimuli? ", "%c", &pair);
651:                          if (toupper(pair)=='Y')
652:                              sprintf(ptr->filename, "%-dB.GRP", ++num_groups);
653:                          else
654:                              sprintf(ptr->filename, "%-d.GRP", ++num_groups);
655:                          }
656:                        for (last=group_ptr; last->next!=NULL && stricmp(ptr->name, (last->next)->name)>0; last=last->next);
657:                        ptr->next = last->next;
658:                        last->next = ptr;
659:                        break;
```

```
660:            case 'D': cprintf("D\n\r\r\r");
662:                      get_input("Enter name of group to delete: ", "", group_ptr->name);
663:                      for (last=group_ptr; last->next!=NULL && stricmp(group_ptr->name, (last->next)->name)!=0; last=last->next);
664:                      if (last->next==NULL && stricmp(group_ptr->name, last->name)!=0)
665:                        print_error("Group not found.");
666:                      else {
667:                        ptr=last->next;
668:                        last->next = ptr->next;
669:                        free(ptr);
670:                        }
671:                      break;
672:            case 'M': cprintf("M\n\r\r\r");
673:                        /* update new group directory file */
674:                      if ((file_in=fopen(GROUP_DIR, "wt"))==NULL) {
675:                        print_system_error();
676:                        for (last=ptr=group_ptr; ptr!=NULL; ) {
677:                            ptr = last->next;
678:                            free(last);
679:                            last = ptr;
680:                            }  /* for */
681:                        return;
682:                      }  /* if */
683:                      fprintf(file_in, "%d\n", num_groups);
684:                      for (ptr=group_ptr->next; ptr!=NULL; ptr=ptr->next)
685:                      fprintf(file_in, "%s\n%s\n", ptr->name, ptr->filename);
686:                      fclose(file_in);
687:                      modify_set_info();
688:                      break;
689:            case 'X': command=ESC_KEY;
690:            case ESC_KEY: cprintf("X");
691:                          break;
692:            }  /* switch */
693:        }  /* for */
694:
695:        /* write new group directory file */
696:    strcpy (temp, EXE_DIR);
697:    strcat (temp, GROUP_DIR);
698:    if ((file_in=fopen(temp, "wt"))==NULL) {
699:        print_system_error();
700:        for (last=ptr=group_ptr; ptr!=NULL; ) {
701:            ptr = last->next;
702:            free(last);
703:            last = ptr;
704:            }  /* for */
705:        return;
706:        }  /* if */
707:    fprintf(file_in, "%d\n", num_groups);
708:    for (ptr=group_ptr->next; ptr!=NULL; ptr=ptr->next)
709:        fprintf(file_in, "%s\n%s\n", ptr->name, ptr->filename);
710:    fclose(file_in);
711:
712:        /* free memory and return to caller */
713:    for (last=ptr=group_ptr; ptr!=NULL; ) {
714:        ptr = last->next;
```

```
715:         free(last);
717:         last = ptr;
718:         }
719:
720:     }   /* modify_group_info */
721: /*-------------------------------------------------------------------------*/
722:
723:
724:
725: /*---------------------------------- ----------------------------------*/
726: void modify_set_info(void)
727:
728:     /*      This module adds, or deletes sets or their entries.     */
729:
730:     {
731:     struct node {
732:         char        filename[60];
733:         int         phase;
734:         struct node *next;
735:         };
736:     typedef struct node pair_entry;
737:
738:     boolean     bandwidth;
739:     char        command;
740:     FILE        *file_in;
741:     int         index;
742:     int         num_freq[MAX_SETS];
743:     int         num_orient[MAX_SETS];
744:     int         num_level[MAX_SETS];
745:     int         page=0;
746:     pair_entry  *pairs_ptr[MAX_SETS];
747:     pair_entry  *ptr;
748:     pair_entry  *last;
749:     int         num;
750:     char        temp1[100];
751:     char        temp2[100];
752:     char        temp3[100];
753:     int         val;
754:
755:
756:         /* get group name and setup screen */
757:     printf("\n\n");
758:     get_input("Enter name of group to modify: ", "", temp1);
759:     print_title("GEXPT 2: Modify Set Info\n\n\n");
760:
761:         /* open group directory file and skip number of groups */
762:     strcpy (temp2, EXE_DIR);
763:     strcat (temp2, GROUP_DIR);
764:     if ((file_in=fopen(temp2, "rt"))==NULL) {
765:         ..int_system_error();
766:         return;
767:         }
768:     fscanf(file_in, "%*d ");
769:
```

```
770:        /* determine filename of group */
772:    for (strcpy(temp2, ""); !feof(file_in) && strcmp(temp1, temp2); ) {
773:        fgets(temp2, 100, file_in);
774:        temp2[strlen(temp2)-1] = '\0';  /* kill CR */
775:        fgets(temp3, 100, file_in);
776:        temp3[strlen(temp3)-1] = '\0';  /* kill CR */
777:        }
778:    fclose(file_in);
779:    if (strcmp(temp1, temp2)) {
780:        print_error("Unable to find specified group.");
781:        return;
782:        }
783:    bandwidth = (toupper(temp3[strlen(temp3)-5]) == 'B');
784:    strcpy (temp1, EXE_DIR);
785:    strcat (temp1, temp3);
786:    strcpy (temp3, temp1);
787:
788:        /* if file exists open pair file and read in number of sets,
789:         * else create a new file */
790:    if (access(temp3, 00)) {
791:        if ((file_in=fopen(temp3, "wt"))==NULL) {
792:            print_system_error();
793:            return;
794:            }
795:        fprintf(file_in, "0\n");
796:        fclose(file_in);
797:        }
798:    if ((file_in=fopen(temp3, "rt"))==NULL) {
799:        print_system_error();
800:        return;
801:        }
802:    fscanf(file_in, "%d ", &num);
803:
804:        /* allocate space for each head pointer and initialize each *
805:         * next pointer                                              */
806:    for (index=0; index<MAX_SETS; index++) {
807:        if ((pairs_ptr[index]=(pair_entry *)malloc(sizeof(pair_entry)))==NULL) {
808:            print_error("Insufficient memory.");
809:            fclose(file_in);
810:            for (; index>=0; index--)
811:                free(pairs_ptr[index]);
812:            return;
813:            }  /* if */
814:        (pairs_ptr[index])->next = NULL;
815:        num_freq[index] = num_orient[index] = num_level[index] = 0;
816:        }  /* for */
817:
818:        /* read in the sets and add the set entry to the linked list of *
819:         * entries                                                       */
820:    for (index=0; index<num && !feof(file_in); index++) {
821:        if (bandwidth)
822:            fscanf(file_in, "%d %d %d\n", &num_orient[index], &num_freq[index], &num_level[index]);
823:        else
824:            fscanf(file_in, "%d %d\n", &num_orient[index], &num_freq[index]);
```

```
825:        for (last=pairs_ptr[index], strcpy(temp1, ""); !feof(file_in) && strcmp(temp1, "."); last=ptr) {
827:            if ((ptr=(pair_entry *)malloc(sizeof(pair_entry)))==NULL) {
828:                print_error("Insufficient memory.");
829:                fclose(file_in);
830:                for (index=0; index<num; index++)
831:                    for (last=ptr=pairs_ptr[index]; ptr!=NULL; ) {
832:                        ptr = last->next;
833:                        free(last);
834:                        last = ptr;
835:                        }  /* for */
836:                return;
837:                }  /* if */
838:            fgets(temp1, 100, file_in);
839:            temp1[strlen(temp1)-1] = '\0';  /* kill CR */
840:            if (strcmp(temp1, ".")) {
841:                sscanf(temp1, "%s %d", ptr->filename, &(ptr->phase));
842:                ptr->next = last->next;
843:                last->next = ptr;
844:                }
845:            }  /* for */
846:        }   /* for */
847:        fclose(file_in);
848:
849:        /* determine number of sets; print sets, menu, and prompt;  *
850:         * execute option                                           */
851:    for (command=' '; command!=ESC_KEY; ) {
852:        for (num=MAX_SETS; num>0 && (pairs_ptr[num-1])->next==NULL; num--);
853:        print_title("GEXPT 2: Modify Set Info\n");
854:        cprintf("%*cCurrent Sets (page %-d of %-d):", ENTRY_INDENT, ' ',
855:                (page/SETS_PER_PAGE+1),
856:                (int)((num+1.0)/SETS_PER_PAGE+1.0-1.0/SETS_PER_PAGE));
857:        if ((pairs_ptr[0])->next==NULL)
858:            cprintf(" none.");
859:        else {
860:            for (index=page; index-page<SETS_PER_PAGE && index<num && index<MAX_SETS; index++) {
861:                if (bandwidth)
862:                    val = command = cprintf("\n\r%*cSET %c (%2d, %2d, %1d): ", ENTRY_INDENT+2, ' ',
863:                                (char)(index+'A'), num_freq[index], num_orient[index], num_level[index])-1;
864:                else
865:                    val = command = cprintf("\n\r%*cSET %c (%2d, %2d): ", ENTRY_INDENT+2, ' ',
866:                                (char)(index+'A'), num_freq[index], num_orient[index])-1;
867:                for (ptr=(pairs_ptr[index])->next; ptr!=NULL; ptr=ptr->next)
868:                    if (command+strlen(ptr->filename)+7>=80)
869:                        command = cprintf("\n\r%*c%s (%1d), ", val, ' ',
870:                                    ptr->filename, ptr->phase)-1;
871:                    else
872:                        command += cprintf("%s (%1d), ", ptr->filename,
873:                                    ptr->phase);
874:                cprintf("\b\b.");
875:                }  /* for */
876:            }  /* else */
877:        printf("\n\n");
878:        print_option("A|Add an image name to a set");
879:        print_option("D|Delete an image name from a set");
```

```
880:          print_option("#|# of page to display");
882:          putch('\n');
883:          print_option("x|Exit menu");
884:          print_option("<ESC>|<ESC> Exit menu");
885:          cprintf("\n\r%*cEnter Option: ", ENTRY_INDENT, ' ');
886:          textcolor(ENTRY_COLOR);
887:          switch(command = toupper(getch())) {
888:              case 'A': cprintf("A\r\n\n\n");
889:                        get_input("Enter letter of set to add to: ", "%c", &command);
890:                        index = toupper(command) - 'A';
891:                        command = 'A';
892:                        if ((pairs_ptr[index])->next==NULL) {
893:                          get_input("Enter number of orientations in image:", "%d", &num_orient[index]);
894:                          get_input("Enter number of frequencies in image:", "%d", &num_freq[index]);
895:                          if (bandwidth)
896:                              get_input("Enter bandwidth level:", "%d", &num_level[index]);
897:                          }
898:                        do {
899:                          if ((ptr=(pair_entry *)malloc(sizeof(pair_entry)))==NULL) {
900:                              print_error("Insufficient memory.");
901:                              break;
902:                              }
903:                          strcpy(ptr->filename, NULL);
904:                          get_input("Enter name of image to add (no extension, CR to end):", "%s", ptr->filename);
905:                          if (!strcmp(ptr->filename, NULL))
906:                              break;
907:                          get_input("Enter phase of image:", "%d", &val);
908:                          ptr->phase = val;
909:                          if (index>MAX_SETS || index<0) {
910:                              print_error("Illegal set.");
911:                              break;
912:                              }
913:                          for (last=pairs_ptr[index]; last->next!=NULL && (ptr->phase > (last->next)->phase); last=last->next);
914:                          ptr->next = last->next;
915:                          last->next = ptr;
916:                          page = (index/SETS_PER_PAGE)*SETS_PER_PAGE;
917:                          } while (strcmp(ptr->filename, NULL));
918:                        break;
919:             case 'D': cprintf("D\n\r\r\r");
920:                        get_input("Enter letter of set to delete from: ", "%c", &command);
921:                        index = toupper(command) - 'A';
922:                        command = 'D';
923:                        do {
924:                          strcpy(temp1, NULL);
925:                          get_input("Enter name of image to delete (no extension, CR to end): ", "%s", temp1);
926:                          if (!strcmp(temp1, NULL))
927:                              break;
928:                          if (index>MAX_SETS || index<0) {
929:                              print_error("Illegal set.");
930:                              break;
931:                              }
932:                          for (last=pairs_ptr[index]; last->next!=NULL && stricmp(temp1, (last->next)->filename); last=last->next);
933:                          if (last->next==NULL && stricmp((pairs_ptr[index])->filename, last->filename))
934:                              print_error("Image not found.");
```

```
                        else {
                            ptr=last->next;
                            last->next = ptr->next;
                            free(ptr);
                            }
                        if ((pairs_ptr[index])->next==NULL)
                            num_orient[index] = num_freq[index] = num_level[index] = 0;
                        page = ((index-1)/SETS_PER_PAGE)*SETS_PER_PAGE;
                        } while (strcmp(temp1, NULL));
                    break;
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9': page = min((command-'1')*SETS_PER_PAGE, num-num%SETS_PER_PAGE);
                      if (page == num)
                        page = page - SETS_PER_PAGE;
                      break;
            case 'X': command=ESC_KEY;
            case ESC_KEY: cprintf("X");
                          break;
            }  /* switch */
        }  /* for */

        /* write new set file */
    if ((file_in=fopen(temp3, "wt"))==NULL) {
        print_system_error();
        for (index=0; index<num; index++)
            for (last=ptr=pairs_ptr[index]; ptr!=NULL; ) {
                ptr = last->next;
                free(last);
                last = ptr;
                }  /* for */
        return;
    }  /* if */
    fprintf(file_in, "%d\n", num);
    for (index=0; index<num; index++) {
        if (bandwidth)
            fprintf(file_in, "%d %d %d\n", num_orient[index], num_freq[index], num_level[index]);
        else
            fprintf(file_in, "%d %d\n", num_orient[index], num_freq[index]);
        for (ptr=(pairs_ptr[index])->next; ptr!=NULL; ptr=ptr->next)
            fprintf(file_in, "%s %d\n", ptr->filename, ptr->phase);
        fprintf(file_in, ".\n");
        }
    fclose(file_in);

        /* free memory and return to caller */
    for (index=0; index<num; index++)
        for (last=ptr=pairs_ptr[index]; ptr!=NULL; ) {
```

```
990:            ptr = last->next;
992:            free(last);
993:            last = ptr;
994:            }
995:
996:    }   /* modify_set_info */
997: /*-------------------------------------------------------------------------*/
```

```
/*****************************************************************************

            FILENAME:               gexpt2a.h
            PROGRAMMER:             Christopher Voltz - UDRI
            CREATED:                -1/8810.18
            LAST MODIFIED:          -1/8904.18
            INTERFACE PROTOCOL:     TURBO C 2.0
            USAGE:                  * this module is intended only for      *
                                    * inclusion with gexpt2 and requires    *
                                    * all the same types, etc. that gexpt2  *
                                    * requires                              *
                                    #include <gexpt2a.h>


        This header file provides prototypes for the following routines:

    ascii_to_image =>   converts an ASCII format image file (.ASC) to an
                        IMAGEACTION format image file (.IMG)
    create_info_file => makes a file containing information necessary to
                        create an image
    create_pairs =>     takes an image (presumably a right image) and makes
                        its left counterpart image so that the two images
                        may be displayed together to form two identical images
                        located equal spaces from the center of the screen
    display_image =>    asks for the filenames of two images and loads them
                        into quadrants 0 and 1 (or full screen if necessary)
    display_texture_menu => displays the options available in the texture
                            menu, processes the input, and calls the correct
                            routine
    extract_image =>    extracts a portion of an image file and writes the
                        ascii version to a file
    flash_pairs =>      reads a lists of pairs from a file and flashes them
                        briefly on the screen
    grab_image =>       digitizes an image and saves it to a user specified
                        file
    image_to_ascii =>   converts an IMAGEACTION format image file (.IMG) to
                        an ASCII format image file (.ASC)
    make_images =>      determines the name of create data file and which
                        type of create is to be done (rectangular or gaussian)
                        and runs the appropriate executable file

*****************************************************************************/



/***********************
 * FUNCTION PROTOTYPES *
 ***********************/

void ascii_to_image(void);
void create_info_file(void);
int  create_pairs(char *filename);
void display_image(void);
byte display_texture_menu(byte *menu);
```

FILE=GEXPT2A.H Fri Jun 16 01:58:16 1989 PAGE=1

```
55: void extract_image(void);
57: void flash_pairs(void);
58: void grab_image(void);
59: void image_to_ascii(void);
60: void make_image_pairs(void);
61: byte make_images(void);
```

```
/*****************************************************************************

            FILENAME:               gexpt2a.c
            PROGRAMMER:             Christopher Voltz - UDRI
            CREATED:                -1/8810.18
            LAST MODIFIED:          -1/8906.13
            INTERFACE PROTOCOL:     TURBO C 2.0
            USAGE:                  #include <gexpt2a.h>


        This include module includes the following routines:

    ascii_to_image =>   converts an ASCII format image file (.ASC) to an
                        IMAGEACTION format image file (.IMG)
    create_info_file => makes a file containing information necessary to
                        create an image
    create_pairs =>     takes an image (presumably a right image) and makes
                        its left counterpart image so that the two images
                        may be displayed together to form two identical images
                        located equal spaces from the center of the screen
    display_image =>    asks for the filenames of two images and loads them
                        into quadrants 0 and 1 (or full screen if necessary)
    display_texture_menu => displays the options available in the texture
                            menu, processes the input, and calls the correct
                            routine
    extract_image =>    extracts a portion of an image file and writes the
                        ascii version to a file
    grab_image =>       digitizes an image and then saves it.
    flash_pairs =>      reads a lists of pairs from a file and flashes them
                        briefly on the screen
    image_to_ascii =>   converts an IMAGEACTION format image file (.IMG) to
                        an ASCII format image file (.ASC)
    make_image_pairs => reads through the component file and calls the
                        create_pairs routine to create image pairs for images
                        which were just generated
    make_images =>      determines the name of create data file and which
                        type of create is to be done (rectangular or gaussian)
                        and runs the appropriate executable file


        DEFINITIONS:

        1) IMAGE DATA FILE: a file of ASCII numbers representing the
    intensity level at a given pixel where the column varies the
    fastest.  This type of file is created by a TXT??.EXP
    program written in NDP FORTRAN 386.  It has the extension .TMP

        2) PICTURE FILE: a file with a header of information and a
    string of binary numbers which represents the intensity of a
    given pixel (as above).  This file is intended to be read by
    the IMAGEACTION framegrabber hardware.

        3) IMAGE FILE: a file with a header of information and a string
    of binary numbers (as above).  This file is also intended to

FILE=GEXPT2A.C    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
 55:     be read by the IMAGEACTION framegrabber hardware.  However,
 57:     its intensity values have been scaled so that each step in
 58:     value corresponds to the same step in actual luminance.
 59:
 60:         4) CONVERSION: the process of taking an image data file and
 61:     converting it to a picture file.
 62:
 63:         5) LINEARIZATION: the process of taking a picture file and
 64:     linearizing it to an image file.
 65:
 66:         6) AOI: area of interest; the part of the image with which we
 67:     are concerned (usually the part containing the gabor image).
 68:
 69:         7) TRANSFORMATION: the process of taking an image data file and
 70:     producing another image data file such that the AOI of the
 71:     first file appears on the opposite side in the second file,
 72:     to produce right and left images.
 73:
 74:
 75:         FILE FORMATS:
 76:
 77:         1) an IMAGE DATA FILE consists of ASCII numbers separated by
 78:     commas and delimited by an EOF mark.  The number of numbers
 79:     will be equal to the width of the image multiplied by the length
 80:     of the image.  In this file, the column varies fastest.
 81:     The extension for this file is .ASC
 82:
 83:         2) an IMAGE FILE consists of a header and a string of binary
 84:     numbers stored in the row-major order as above.  Each binary
 85:     number is 2 bytes long and in standard IEEE format.  The file
 86:     is delimited by an EOF mark.  The extension for this file is
 87:     .IMG
 88:
 89: ******************************************************************************/
 90:
 91:
 92:
 93: /****************
 94:  * HEADER FILES *
 95:  ****************/
 96:
 97:     /*** standard TURBO C header files ***/
 98: #include <alloc.h>
 99: #include <conio.h>
100: #include <ctype.h>
101: #include <dos.h>
102: #include <math.h>
103: #include <stdio.h>
104: #include <stdlib.h>
105: #include <string.h>
106: #include <time.h>
107:
108:     /*** program specific header files ***/
109: #include <constant.h>           /* program constants to define system   */

FILE=GEXPT2A.C    Fri Jun 16 01:58:16 1989    PAGE=2
```

```
110:                                    /* parameters for included files       */
112: #if DT2871
113: #    include <dt2871.h>            /* routines to control DT-2871 board   */
114: #else
115: #    include <pcwrap.h>            /* wrapper for pcvision routines       */
116: #endif
117: #include <toolbox.h>               /* general utility routines            */
118: #include <gexpt2a.h>               /* this module's header file           */
119: #include <gexpt2b.h>               /* header file of collection routines  */
120:
121:
122:
123: /************************
124:  * FUNCTION DEFINITIONS *
125:  ************************/
126:
127: /*-------------------------------------------------------------------------*/
128: void ascii_to_image(void)
129:
130:     /*      This module converts an IMAGEACTION format file to an
131:         ASCII format file.                                                 */
132:
133:     {
134:     char            filename[256];      /* name of file to convert      */
135:     FILE            *file_in;           /* pointer for input file       */
136:     FILE            *file_out;          /* pointer for output file      */
137:     header_type     header;             /* header for image             */
138:     unsigned long   size;               /* number of bytes in image     */
139:     char            temp[100];          /* temporary string             */
140:     int             val;                /* value read in                */
141:
142:
143:         /*** setup screen ***/
144:     print_title("GEXPT 2: Convert ASCII to Image\n\n");
145:
146:         /*** get filename of file to convert and then open input file ***/
147:     get_input("Enter name of ascii file to convert (no extension):", "%s", filename);
148:     if (filename[0]==EXPAND_CHAR) {
149:         strcpy(temp, IMAGE_DIR);
150:         strcat(temp, filename+1);
151:         strcpy(filename, temp);
152:         }
153:     strcat(filename, ".ASC");
154:     file_in = fopen(filename, "rt");
155:     if (file_in==NULL) {
156:         print_system_error();
157:         return;
158:         }
159:
160:         /*** determine name of output file and then open output file ***/
161:     filename[strlen(filename)-3] = '\0';
162:     strcat(filename, "IMG");
163:     file_out = fopen(filename, "wb");
164:     if (file_out==NULL) {
```

```
165:         print_system_error();
167:         return;
168:         }
169:
170:         /*** make header ***/
171:     size = 256;
172:     get_input("Enter size of image: <256>", "%d", &size);
173:     textcolor(MENU_COLOR);
174:     header[0] = 'I';
175:     header[1] = 'M';
176:     header[4] = header[6] = (byte) (size % 256);
177:     header[5] = header[7] = (byte) (size / 256);
178:     header[8] = header[9] = header[10] = header[11] = header[12] = header[13] = 0;
179:     header[64] = ' ';
180:     header[65] = '\0';
181:     get_input("Enter comment (255 chars):", "", &header[64]);
182:     if ((val=strlen((char *)&header[64]))>255) {
183:         print_error("Comment too long.  Program corrupted.");
184:         abort();
185:         }
186:     else {
187:         if (val==0) {
188:             header[2] = 1;
189:             header[3] = 0;
190:             }
191:         else {
192:             header[2] = (byte) (val % 256);
193:             header[3] = 0;
194:             }
195:         }
196:     for (val=14; val<64; header[val++] = 0);
197:
198:         /*** write header ***/
199:     for (val=0; val<=header[2]+63; val++)
200:         fprintf(file_out, "%c", (char) header[val]);
201:
202:         /*** process file ***/
203:     while (!feof(file_in)) {
204:         fscanf(file_in, " %d", &val);
205:         if (!feof(file_in))
206:             fprintf(file_out, "%c", (char) val);
207:         }
208:
209:     fclose(file_in);
210:     fclose(file_out);
211:
212:     }   /* ascii_to_image */
213: /*-------------------------------------------------------------------------*/
214:
215:
216:
217: /*-------------------------------------------------------------------------*/
218: void create_info_file(void)
219:
```

```
220:    /*      This module reads in the data required to generate an image
222:        and writes it out to a file in the proper format so it can be
223:        read later by the routines to create the images.                */
224:
225:    {
226:    enum        {BANDWIDTH, EVEN_ODD, NONE, NORMAL};
227:    const float PI=3.141592654;
228:
229:    float   center_x;
230:    float   center_y;
231:    char    filename[80];
232:    FILE    *file_out;
233:    float   freq_angle_inc;
234:    float   freq_angle_offset;
235:    float   ftemp;
236:    char    image_type;
237:    int     index;
238:    int     index_2;
239:    float   magnitude;
240:    int     num_components;
241:    int     num_freq;
242:    int     num_levels;
243:    char    step_type;
244:    float   width;
245:    int     random_type;
246:    int     random_nums[MAX_COMPONENTS][2];
247:    char    temp[100];
248:    int     val;
249:
250:
25?:    print_title("GEXPT 2: Create Info File\n\n");
252:
253:    get_input("Enter data filename (no extension, eg. GxxxGyyy):", "%s", filename);
254:    if (filename[0]==EXPAND_CHAR) {
255:        strcpy(temp, DATA_DIR);
256:        strcat(temp, filename+1);
257:        strcpy(filename, temp);
258:        }
259:    strcat(filename, ".DAT");
260:
261:    cprintf("\r\n");
262:    get_input("Enter type of image to be created (<r>=rectangular, g=gaussian):", "%c", &image_type);
263:    if (image_type!='g' && image_type!='r')
264:        image_type = IMAGE_TYPE_DEFAULT;
265:    image_type = toupper(image_type);
266:
267:    if ((file_out=fopen(filename, "wt"))==NULL) {
268:        print_system_error();
269:        return;
270:        }
271:
272:    do {
273:        clrscr();
274:        textcolor(MENU_COLOR);
```

```
275:        cprintf("When done entering images press return in response to the image name prompt.\n\r\n\r");
277:        filename[0] = 0;
278:        get_input("Enter image name (CR to end):", "%s", filename);
279:        if (strlen(filename) == 0)
280:            break;
281:        fprintf(file_out, "'%s'\n", filename);
282:        printf("\n");
283:
284:        get_input("Enter number of components:", "%d", &num_components);
285:        cprintf("\r\n");
286:
287:        sprintf(filename, "Enter image size <%-d>:", SIZE_DEFAULT);
288:        val = SIZE_DEFAULT;
289:        get_input(filename, "%d", &val);
290:        fprintf(file_out, "%d\n", val);
291:
292:        fprintf(file_out, "%d\n", num_components);
293:
294:        sprintf(filename, "Enter mean luminance <%-d>:", LUM_DEFAULT);
295:        val = LUM_DEFAULT;
296:        get_input(filename, "%d", &val);
297:        fprintf(file_out, "%d\n", val);
298:
299:        sprintf(filename, "Enter maximum luminance <%-d>:", MAX_LUM_DEFAULT);
300:        val = MAX_LUM_DEFAULT;
301:        get_input(filename, "%d", &val);
302:        fprintf(file_out, "%d\n", val);
303:        printf("\n");
304:
305:        sprintf(filename, "Enter x,y coords of center of image <%-.2f, %-.2f>:",
306:                CENTER_X_DEFAULT, CENTER_Y_DEFAULT);
307:        center_x = CENTER_X_DEFAULT;
308:        center_y = CENTER_Y_DEFAULT;
309:        get_input(filename, "%f,%f", &center_x, &center_y);
310:
311:        get_input("Do you wish to randomize the phase <y>:", "%c", &filename[0]);
312:        if (toupper(filename[0])!='N') {
313:            get_input("    Randomization type -- (N)ormal, (E)ven/Odd, or (B)andwith <N>:", "%c", &filename[0]);
314:            switch (toupper(filename[0])) {
315:                case 'E': random_type = EVEN_ODD;
316:                          break;
317:                case 'B': random_type = BANDWIDTH;
318:                          break;
319:                default:  random_type = NORMAL;
320:                }   /* switch */
321:            num_levels = num_components;
322:            if (random_type!=BANDWIDTH)
323:                get_input("    Enter number of random levels <num_components>:", "%d", &num_levels);
324:            else
325:                get_input("    Enter bandwidth (1/range):", "%d", &num_levels);
326:            switch (random_type) {
327:                case NORMAL: for (index=0; index<num_components; index++)
328:                                 random_nums[index][0] = index%num_levels+1;
329:                             val = 1;
```

```
330:                            break;
332:                case EVEN_ODD: for (index=0; index<2; index++)
333:                                 for (index_2=0; index_2<num_components/2; index_2++)
334:                                     random_nums[index_2][index] = index_2%num_levels+1;
335:                               val = 2;
336:                               break;
337:                case BANDWIDTH: for (index=0; index<2; index++)
338:                                  for (index_2=0; index_2<num_components/2; index_2++)
339:                                     random_nums[index_2][index] = index_2;
340:                               val = 2;
341:                }   /* switch */
342:            }   /* if then */
343:        else {
344:            random_type = NONE;
345:            num_levels = 1;
346:            val = 0;
347:            }   /* if else */
348:
349:        randomize();
350:
351:        for (index=0; index<val; index++)
352:            for (index_2=0; index_2<num_components/val; index_2++)
353:                swap_int(&random_nums[index_2][index],
354:                         &random_nums[random(num_components/val)][index]);
355:
356:        sprintf(filename, "Enter magnitude of components <%-.2f>:", MAG_DEFAULT);
357:        magnitude = MAG_DEFAULT;
358:        get_input(filename, "%f", &magnitude);
359:
360:        sprintf(filename, "Enter width of image <%-.2f>:", WIDTH_DEFAULT);
361:        width = WIDTH_DEFAULT;
362:        get_input(filename, "%f", &width);
363:        cprintf("\n\r");
364:
365:        get_input("Enter type of step: (<l>=logarithmic, s=standard):", "%c", &step_type);
366:        if (step_type!='l' && step_type!='s')
367:            step_type = STEP_TYPE_DEFAULT;
368:        step_type = toupper(step_type);
369:
370:        get_input("Enter number of spatial frequencies:", "%d", &num_freq);
371:        freq_angle_inc = 180.0 / ((float)num_components / num_freq);
372:        freq_angle_offset = 0.0;
373:
374:        for (index=0; index<num_freq; index++)
375:            for (val=0; val<num_components/num_freq; val++) {
376:                switch (random_type) {
377:                    case NONE: ftemp = 0.0;
378:                               break;
379:                    case NORMAL: ftemp=random_nums[index*num_components/num_freq+val][0]
380:                                       *2*PI/num_levels;
381:                                 break;
382:                    case EVEN_ODD: ftemp=random_nums[index/2*num_components/num_freq+val]
383:                                         [index%2]*2*PI/num_levels+(index%2)*PI;
384:                                   break;
```

```
385:                    case BANDWIDTH: ftemp=random_nums[index/2*num_components/num_freq+val]
387:                                        [index%2]*PI/((num_components/2-1)*num_levels/2)-PI/num_levels+
388:                                        (index%2)*PI;
389:                                break;
390:                    }   /* switch */
391:                if (step_type=='L')
392:                    fprintf(file_out, "%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f\n",
393:                        center_x, center_y,
394:                        pow(12.0, index/(num_freq-1.0))*((image_type=='R') ? 1.0 : 1.25/1.7),
395:                        freq_angle_offset+val*freq_angle_inc, magnitude, ftemp, width);
396:                else
397:                    fprintf(file_out, "%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f\n",
398:                        center_x, center_y,
399:                        (1.0+index*11.0/(num_freq-1))*((image_type=='R') ? 1.0 : 1.25/1.7),
400:                        freq_angle_offset+val*freq_angle_inc, magnitude, ftemp, width);
401:                }
402:
403:    } while (strlen(filename) != 0);
404:
405:    fclose(file_out);
406:
407:    }   /* create_info_f le */
408: /*-------------------------------------------------------------------------*/
409:
410:
411:
412: /*-------------------------------------------------------------------------*/
413: int create_pairs(char *filename)
414:
415:    /*      This module reads in an image (presumably a right image) and
416:        creates a file containing a counterpart image (a left image) which
417:        is created so that when the two images are displayed together as
418:        a pair, their centers will be located equal distances away from the
419:        center of the screen and the two images will be identical.  It
420:        assumes a square image and that the last pixel of the first line is
421:        of the background intensity.                                        */
422:
423:    {
424:    char        command[255];
425:    int         end_y;
426:    FILE        *file_in;
427:    FILE        *file_out;
428:    header_type header;
429:    char huge   *image;
430:    unsigned    size;
431:    int         start_y;
432:    int         x;
433:    int         y;
434:
435:
436:        /*** allocate array on heap ***/
437:    image = (void huge *)farmalloc((long)MAX_IMAGE_SIZE*(long)MAX_IMAGE_SIZE);
438:    if (image==NULL) {
439:        sprintf(command, "Insufficient memory (%-ld bytes available"
```

```
                            " %-ld bytes required)", farcoreleft(),
                            (long)MAX_IMAGE_SIZE*(long)MAX_IMAGE_SIZE);
        print_error(command);
        return(1);
        }

        /*** display filename ***/
    textcolor(MENU_COLOR);
    cprintf("%*cCreating left/right pair from: %s.\n\r", ENTRY_INDENT, ' ', filename);

        /*** append L to original filename and then open file ***/
    sprintf(command, "COPY %s.IMG %s%sL.IMG", filename, IMAGE_DIR, filename);
    if (system(command)!=0) {
        print_error("Unable to create left version of image.  Check disk.");
        farfree((void far *)image);
        return(2);
        }
    sprintf(command, "ERASE %s.IMG", filename);
    system(command);
    sprintf(command, "%s%sL.img", IMAGE_DIR, filename);

    if ((file_in=fopen(command, "rb"))==NULL) {
        print_system_error();
        farfree((void far *)image);
        return(2);
        }

        /*** determine name of output file and then open output file ***/
    command[strlen(command)-5] = 'R';
    file_out = fopen(command, "wb");
    if (file_out==NULL) {
        print_system_error();
        fclose(file_in);
        farfree((void far *)image);
        return(2);
        }

        /*** read header ***/
    fread(header, sizeof(byte), 64, file_in);
    fread(&header[64], sizeof(byte), header[2]+256*header[3], file_in);
    size = header[4]+256*header[5];

        /*** write header ***/
    for (x=0; x<64+header[2]+256*header[3]; x++)
        fprintf(file_out, "%c", header[x]);

        /*** read the image into the array ***/
    for (x=0; x<size; x++)
        for (y=0; y<size; y++)
            fscanf(file_in, "%c", (image+x*(long)MAX_IMAGE_SIZE+y));

        /*** determine start and ending y of image ***/
    for (y=0, start_y=TRUE; y<size && start_y; y++)
        for (x=0; x<size && start_y; x++)
```

```
            start_y = *(image+x*(long)MAX_IMAGE_SIZE+y)==*image;
    start_y = y;

    for (y=size-1, end_y=TRUE; y>=0 && end_y; y--)
        for (x=size-1; x>=0 && end_y; x--)
            end_y = *(image+x*(long)MAX_IMAGE_SIZE+y)==*image;
    end_y = y;

        /*** write out image ***/
    for (x=0; x<size; x++) {
        for (y=size-1; y>end_y; y--)
            fprintf(file_out, "%c", *image);
        for (y=start_y; y<=end_y; y++)
            fprintf(file_out, "%c", *(image+x*(long)MAX_IMAGE_SIZE+y));
        for (y=start_y-1; y>=0; y--)
            fprintf(file_out, "%c", *image);
        }

        /*** close the files, free allocated memory, and return to ***
         *** texture menu                                          ***/
    fclose(file_out);
    fclose(file_in);
    farfree((void far *)image);
    return(0);
    }   /* create_pairs */
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
void display_image(void)

    /*      This module requests the name of a pair of images and displays
        them in quadrants 0 and 1.  Control is returned to the caller when
        the user presses a key.                                          */

    {
    char    filename[60];   /* filename of image        */
    header_type header;     /* image header             */
    char    temp[100];      /* temporary string         */


        /*** display menu info ***/
    print_title("GEXPT 2: Display Image\n\n");

        /*** display first image ***/
    filename[0] = '\0';
    get_input("Enter name of first image (no extension):", "%s", filename);
    textcolor(MENU_COLOR);
    cprintf("\n\r");
    if (filename[0]==EXPAND_CHAR && filename[1]==EXPAND_CHAR) {
        sprintf(temp, "%sg%sgl", IMAGE_DIR, filename+2);
        strcpy(filename, temp);
        }
```

```
550:     else if (filename[0]==EXPAND_CHAR) {
552:         strcpy(temp, IMAGE_DIR);
553:         strcat(temp, filename+1);
554:         strcpy(filename, temp);
555:         }
556:     display_buffer(SIGNAL_BUFFER);
557:     if (filename[0]) {
558:         errno = 0;
559:         strcat(filename, ".IMG");
560:         read_image(SIGNAL_BUFFER, LEFT, filename, header);
561:         if (errno == 0)
562:             print_header_info(header);
563:         }
564:
565:         /*** display second image ***/
566:     cprintf("\n\r");
567:     filename[0] = '\0';
568:     get_input("Enter name of second image (no extension):", "%s", filename);
569:     textcolor(MENU_COLOR);
570:     cprintf("\n\r");
571:     if (filename[0]==EXPAND_CHAR && filename[1]==EXPAND_CHAR) {
572:         sprintf(temp, "%sg%sgr", IMAGE_DIR, filename+2);
573:         strcpy(filename, temp);
574:         }
575:     else if (filename[0] == EXPAND_CHAR) {
576:         strcpy(temp, IMAGE_DIR);
577:         strcat(temp, filename+1);
578:         strcpy(filename, temp);
579:         }
580:     if (filename[0]) {
581:         errno = 0;
582:         strcat(filename, ".IMG");
583:         read_image(SIGNAL_BUFFER, RIGHT, filename, header);
584:         if (errno == 0)
585:             print_header_info(header);
586:         }
587:     textcolor(ERROR_COLOR);
588:     cprintf("\n\rPRESS ANY KEY TO RETURN TO MAIN\n\r");
589:     getch();
590:     textcolor(MENU_COLOR);
591:
592:     }   /* display_image */
593: /*-------------------------------------------------------------------------*/
594:
595:
596:
597: /*-------------------------------------------------------------------------*/
598: byte display_texture_menu(byte *menu)
599:
600:     /*      This module displays the texture generation/presentation
601:         menu, gets a keystroke, executes the appropriate function, and
602:         returns either a non-zero value if the menu should be displayed
603:         again, or a zero value if the user selected the exit option.    */
604:
```

```
605:    {
607:    char    command[100];   /* command to execute in DOS    */
608:    char    mask[60];       /* mask for directory operation */
609:
610:
611:        /*** if reentering program after a create, check if pairs ***
612:         *** are to be created.                                     ***/
613:    if (*menu==TEXTURE_MENU) {
614:        make_image_pairs();
615:        *menu = DEFAULT_MENU;
616:        }
617:
618:        /*** setup the screen ***/
619:    print_title("GEXPT 2: Texture Generation/Presentation Menu\n\n");
620:
621:    print_option("C|Create image component info file");
622:    print_option("M|Make textured images");
623:    print_option("p|Convert image to left/right pair");
624:    print_option("i|Convert image file to ASCII file");
625:    print_option("A|Convert ASCII file to image file");
626:    print_option("F|Flash image pairs");
627:    print_option("D|Display one pair of images");
628:    print_option("G|Grab (digitize) an image and save it");
629:    print_option("E|Extract a subimage from an image");
630:    print_option("L|List files");
631:    print_option("S|Shell to DOS");
632:    printf("\n");
633:    print_option("x|Exit menu");
634:    print_option("<ESC>|<ESC> Exit menu");
635:    cprintf("\n\r\n%*cEnter Option: ", ENTRY_INDENT, ' ');
636:
637:        /*** get the user's option ***/
638:    textcolor(ENTRY_COLOR);
639:    switch (toupper(getche())) {
640:        case 'A': ascii_to_image();
641:                  break;
642:        case 'C': create_info_file();
643:                  break;
644:        case 'D': display_image();
645:                  break;
646:        case 'E': extract_image();
647:                  break;
648:        case 'F': flash_pairs()
649:                  break;
650:        case 'G': grab_image();
651:                  break;
652:        case 'I': image_to_ascii();
653:                  break;
654:        case 'L': cprintf("\r\n\n\n");
655:                  get_input("Enter mask [*.*]: ", "", mask);
656:                  if (mask[0]==EXPAND_CHAR)
657:                    sprintf(command, "ddir %s%s", IMAGE_DIR, mask+1);
658:                  else
659:                    sprintf(command, "ddir %s", mask);
```

```
660:                    textcolor(MENU_COLOR);
662:                    clrscr();
663:                    system(command);
664:                    cprintf("\n\rPress any key to continue\n\r");
665:                    textcolor(ENTRY_COLOR);
666:                    getch();
667:                    break;
668:            case 'M': return(make_images());
669:            case 'P': print_title("GEXPT 2: CREATE PAIRS SCREEN\n\n");
670:                    get_input("Enter name of image to convert (no extension):",
671:                              "%s", command);
672:                    create_pairs(command);
673:                    break;
674:            case 'S': textcolor(MENU_COLOR);
675:                    cprintf("\n\r\n\n");
676:                    get_input("Enter command or <enter> to shell:", "", &command[0]);
677:                    textcolor(MENU_COLOR);
678:                    clrscr();
679:                    system(command);
680:                    textcolor(MENU_COLOR);
681:                    cprintf("\n\rPress any key to continue\n\r");
682:                    getch();
683:                    break;
684:            case ESC_KEY:   cprintf("\bX");
685:            case'X': return(EXIT_MENU);             /* exit this menu */
686:            default: cprintf("\a");
687:                    break;
688:            }   /* switch */
689:
690:            return(REDISPLAY);      /* redisplay menu */
691:
692:        }   /* display_texture_menu */
693: /*---------------------------------------------------------------------------*/
694:
695:
696:
697: /*---------------------------------------------------------------------------*/
698: void extract_image(void)
699:
700:     /*      This module extracts a 32x32 pixel matrix from the given
701:         image.  The center of the matrix is specified by the user and
702:         defaults to the center of a created image.                          */
703:
704:     {
705:     float   center_x;           /* x center of matrix           */
706:     float   center_y;           /* y center of matrix           */
707:     int     count;              /* # of #'s per line            */
708:     char    filename[100];      /* name of file to convert      */
709:     FILE    *file_in;           /* pointer for input file       */
710:     FILE    *file_out;          /* pointer for output file      */
711:     header_type header;         /* header for image             */
712:     int     index;              /* general loop variable        */
713:     int     index_2;            /* general loop variable        */
714:     int     size;               /* size of image in pixels      */
```

```
715:     char    temp[100];          /* temporary string             */
717:     byte    val;                /* value read in                */
718:
719:
720:         /*** setup screen ***/
721:     print_title("GEXPT 2: Extract Image\n\n");
722:
723:         /*** get filename of file to analyze and then open input file ***/
724:     get_input("Enter name of image to analyze (no extension):", "%s", filename);
725:     if (filename[0]==EXPAND_CHAR) {
726:         strcpy(temp, IMAGE_DIR);
727:         strcat(temp, filename+1);
728:         strcpy(filename, temp);
729:         }
730:     strcat(filename, ".IMG");
731:     file_in = fopen(filename, "rb");
732:     if (file_in==NULL) {
733:         print_system_error();
734:         return;
735:         }
736:
737:         /*** determine name of output file and then open output file ***/
738:     filename[strlen(filename)-3] = '\0';
739:     strcat(filename, "ASC");
740:     file_out = fopen(filename, "wt");
741:     if (file_out==NULL) {
742:         print_system_error();
743:         return;
744:         }
745:
746:         /*** get center of matrix in user coordinates ***/
747:     center_x = CENTER_X_DEFAULT;
748:     center_y = CENTER_Y_DEFAULT;
749:     sprintf(filename, "Enter x,y center of image <%1.1f, %1.1f>:", center_x, center_y);
750:     get_input(filename, "%f,%f", &center_x, &center_y);
751:
752:         /*** read header ***/
753:     errno = 0;
754:     fread(header, sizeof(byte), 64, file_in);
755:     fread(&header[64], sizeof(byte), header[2]+256*header[3], file_in);
756:
757:         /*** determine size of image ***/
758:     size = header[4]+256*header[5];
759:
760:         /*** calculate center of matrix in pixels ***/
761:     center_x = size * center_x / SIZE_RANGE;
762:     center_y = size * center_y / SIZE_RANGE;
763:
764:         /*** process file: get to proper spot in file ***/
765:     for (index=0; index<center_x-EXTRACT_SIZE/2 && !feof(file_in); index++)
766:         for (index_2=0; index_2<size; index_2++)
767:             fscanf(file_in, "%c", &val);
768:
769:         /*** process file: write data ***/
```

```
770:     for (index=0, count=0; index<EXTRACT_SIZE && !feof(file_in); index++) {
772:         for (index_2=0; index_2<center_y-EXTRACT_SIZE/2 && !feof(file_in); index_2++)
773:             fscanf(file_in, "%c", &val);
774:         for (index_2=0; index_2<EXTRACT_SIZE && !feof(file_in); index_2++) {
775:             fscanf(file_in, "%c", &val);
776:             fprintf(file_out, "%4d", val);
777:             if (count++>31) {
778:                 fprintf(file_out, "\n");
779:                 count = 0;
780:                 }
781:             }   /* next pixel */
782:         for (index_2=center_y+EXTRACT_SIZE/2; index_2<size; index_2++)
783:             fscanf(file_in, "%c", &val);
784:         }   /* next line */
785:
786:     fclose(file_in);
787:     fclose(file_out);
788:
789:     }   /* extract_image */
790: /*-------------------------------------------------------------------------*/
791:
792:
793:
794: /*-------------------------------------------------------------------------*/
795: void flash_pairs(void)
796:     {
797:     float       duration;
798:     char        filenames[2][60];
799:     FILE        *file_out;
800:     FILE        *fileptr;
801:     header_type header;
802:     int         response;
803:     char        temp[60];
804:
805:
806:     print_title("GEXPT 2: Flash Pairs\n\n");
807:
808:     get_input("Enter name of output file:", "%s", filenames[0]);
809:     if (filenames[0][0]==EXPAND_CHAR) {
810:         strcpy(temp, DATA_DIR);
811:         strcat(temp, filenames[0]+1);
812:         strcpy(filenames[0], temp);
813:         }
814:     if ((file_out=fopen(filenames[0], "wt"))==NULL) {
815:         print_system_error();
816:         return;
817:         }
818:
819:     get_input("Enter filename of pairs:", "%s", filenames[0]);
820:     if (filenames[0][0]==EXPAND_CHAR) {
821:         strcpy(temp, DATA_DIR);
822:         strcat(temp, filenames[0]+1);
823:         strcpy(filenames[0], temp);
824:         }
```

```
825:     if ((fileptr=fopen(filenames[0], "rt"))==NULL) {
827:         print_system_error();
828:         return;
829:         }
830:
831:     get_input("Enter duration of stimulus (in ms):", "%f", &duration);
832:     duration = (int) (duration / 16.667);   /* get number of fields */
833:     if (duration < 1.0)
834:         duration = 1.0;
835:
836:     print_title("GEXPT 2: Flash Pairs\n\n");
837:
838:     display_buffer(NOISE_BUFFER);
839:     for (response=11; response!=0 && !feof(fileptr);) {
840:         fscanf(fileptr, "%s %s", filenames[LEFT], filenames[RIGHT]);
841:         if (filenames[LEFT][0] == EXPAND_CHAR) {
842:             strcpy(temp, IMAGE_DIR);
843:             strcat(temp, filenames[LEFT]+1);
844:             strcpy(filenames[LEFT], temp);
845:             }
846:         if (filenames[RIGHT][0] == EXPAND_CHAR) {
847:             strcpy(temp, IMAGE_DIR);
848:             strcat(temp, filenames[RIGHT]+1);
849:             strcpy(filenames[RIGHT], temp);
850:             }
851:         strcat(filenames[LEFT], ".IMG");
852:         strcat(filenames[RIGHT], ".IMG");
853:         if feof(fileptr) continue;
854:         display_buffer(NOISE_BUFFER);
855:         read_image(SIGNAL_BUFFER, LEFT, filenames[LEFT], header);
856:         read_image(SIGNAL_BUFFER, RIGHT, filenames[RIGHT], header);
857:         textcolor(MENU_COLOR);
858:         cprintf("%*cPress enter to flash next pair.\n\r", ENTRY_INDENT, ' ');
859:         getch();
860:         flash_screens((int) 0, (int) duration);
861:         for (response=11; response<0 || response>7;) {
862:             textcolor(MENU_COLOR);
863:             cprintf("%*cEnter response (0 to exit):", ENTRY_INDENT, ' ');
864:             textcolor(ENTRY_COLOR);
865:             response = getche() - '0';
866:             if (response<0 || response>7) {
867:                 textcolor(ERROR_COLOR);
868:                 cprintf("\a%*cIllegal input.  Try again. .\n\r", ENTRY_INDENT, ' ');
869:                 }
870:             else if (response>0 && response<7)
871:                     fprintf(file_out, "%s %s %d\n", filenames[LEFT], filenames[RIGHT], response);
872:             }
873:         cprintf("\n\r");
874:         }
875:
876:     fclose(fileptr);
877:     fclose(file_out);
878:
879:     }   /* flash_pairs */
```

```
880: /*------------------------------------------------------------------------*/
882:
883:
884:
885: /*------------------------------------------------------------ ----------*/
886: void grab_image(void)
887:
888:     /*      This module digitizes an image and then saves it to a user
889:         specified file.                                                  */
890:
891:     {
892:     char    filename[100];
893:     char    temp[100];
894:     int     val;
895:
896:
897: #if DT2871
898:         print_title("GEXPT 2: Grab Image\n\n");
899:         cprintf("Not implemented, yet, for DT-2871.  Sorry.\n\r");
900:         cprintf("Press any key to exit.\n\r");
901:         getch();
902: #else
903:         for (val=CR; val==CR; )
904:         {
905:             /*** setup screen ***/
906:             print_title("GEXPT 2: Grab Image\n\n");
907:
908:             /*** goto grab mode ***/
909:             Set_lut(SIGNAL_BUFFER, INPUT_TABLE);
910:             grab_mode();
911:             Set_lut(SIGNAL_BUFFER, GREEN_TABLE);
912:
913:             /*** wait for user to press key to stop grabbing ***/
914:             cprintf("Press any key to snap image (stop digitizing).\r\n\n\n");
915:             getch();
916:             freeze_mode();
917:
918:             cprintf("(Select 4 to save full screen, 0-3 to save a specific quadrant,\r\n");
919:             cprintf(" or nothing to not save image)\r\n\n");
920:             val = 5;
921:             get_input("Enter quadrant to save:", "%d", &val);
922:             if (val>-1  && val<5)
923:             {
924:                 strcpy(filename, "test.img");
925:                 get_input("Enter pathname for image (no extension):", "%s", filename);
926:                 if (filename[0]==EXPAND_CHAR) {
927:                     strcpy(temp, IMAGE_DIR);
928:                     strcat(temp, filename+1);
929:                     strcpy(filename, temp);
930:                 }
931:                 strcat(filename, ".IMG");
932:                 save_image(SIGNAL_BUFFER, val, filename);
933:             }
934:
```

```
935:                /*** see if we want another image ***/
937:              cprintf("\r\n\nPress ENTER to grab another image, or any other key to exit.\n\r");
938:              val = getch();
939:          }
940: #endif
941:
942:      }   /*  grab_image  */
943: /*--------------------------------------------------------------------------*/
944:
945:
946:
947: /*--------------------------------------------------------------------------*/
948: void image_to_ascii(void)
949:
950:      /*      This module reads in an IMAGEACTION format file and writes
951:          out the corresponding ASCII format file.                        */
952:
953:      {
954:      byte    count;              /* # of #s written on one line  */
955:      char    filename[60];       /* name of file to convert      */
956:      FILE    *file_in;           /* pointer for input file       */
957:      FILE    *file_out;          /* pointer for output file      */
958:      header_type header;         /* header for image             */
959:      char    temp[100];          /* temporary string             */
960:      byte    val;                /* value read in                */
961:
962:
963:          /*** setup screen ***/
964:      print_title("GEXPT 2: Convert Image\n\n");
965:
966:          /*** get filename of file to convert and then open input file ***/
967:      get_input("Enter name of image to convert (no extension):", "%s", filename);
968:      if (filename[0]==EXPAND_CHAR) {
969:          strcpy(temp, IMAGE_DIR);
970:          strcat(temp, filename+1);
971:          strcpy(filename, temp);
972:          }
973:      strcat(filename, ".IMG");
974:      file_in = fopen(filename, "rb");
975:      if (file_in==NULL) {
976:          print_system_error();
977:          return;
978:          }
979:
980:          /*** determine name of output file and then open output file ***/
981:      filename[strlen(filename)-3] = '\0';
982:      strcat(filename, "ASC");
983:      file_out = fopen(filename, "wt");
984:      if (file_out==NULL) {
985:          print_system_error();
986:          return;
987:          }
988
989:          /*** read header ***/
```

```
 990:     fread(header, sizeof(byte), 64, file_in);
 992:     fread(&header[64], sizeof(byte), header[2]+256*header[3], file_in);
 993:
 994:         /*** process file ***/
 995:     while (!feof(file_in)) {
 996:         for (count=0; count<16 && !feof(file_in); count++) {
 997:             fscanf(file_in, "%c", &val);
 998:             if (!feof(file_in))
 999:                 fprintf(file_out, " %3.3d", val);
1000:             }
1001:         fprintf(file_out, "\n");
1002:         }
1003:
1004:     fclose(file_in);
1005:     fclose(file_out);
1006:
1007:     }   /* image_to_ascii */
1008: /*-------------------------------------------------------------------------*/
1009:
1010:
1011:
1012: /*-------------------------------------------------------------------------*/
1013: void make_image_pairs(void)
1014:
1015:     /*      This module reads through the component data file to
1016:         determine image filenames and calls the make pairs procedure
1017:         with those names.  Additionally, it removes the lock file.      */
1018:
1019:     {
1020:     char    filename[100];
1021:     FILE    *file_ptr;
1022:     char    image_type;
1023:     int     num_components;
1024:
1025:
1026:         /*** setup screen ***/
1027:     print_title("GEXPT 2: Create Pairs\n\n");
1028:
1029:         /*** open lock file and determine if pairs were to be generated ***
1030:          *** if they were not, delete the lock file and return to the   ***
1031:          *** texture menu; otherwise, skip the summed/individual info   ***
1032:          *** and read the image type and then close the file             ***/
1033:         if ((file_ptr=fopen(LOCK_FILE, "rt"))==NULL) {
1034:             print_error("Unable to open lock file.  Check disk.");
1035:             return;
1036:             }
1037:         fscanf(file_ptr, "%c", filename);
1038:         if (toupper(filename[0])!='Y') {
1039:             fclose(file_ptr);
1040:             if (unlink(LOCK_FILE)!=0)
1041:                 print_error("Unable to remove lock file.  Check disk.");
1042:
1043:             return;
1044:             }
```

```
1045:         fscanf(file_ptr, "%*c%c", &image_type);
1047:         fclose(file_ptr);
1048:
1049:         /*** open component data file ***/
1050:     if ((file_ptr=fopen(CREATE_FILE, "rt"))==NULL) {
1051:         print_error("Unable to open " CREATE_FILE ".  Check disk.");
1052:         if (unlink(LOCK_FILE)!=0)
1053:             print_error("Unable to remove lock file.  Check disk.");
1054:         return;
1055:         }
1056:
1057:         /*** read a filename, create the pairs, skip to the next ***
1058:          *** filename and continue until eof                     ***/
1059:     for (fscanf(file_ptr, "%s", filename); !feof(file_ptr); fscanf(file_ptr, "%s", filename)) {
1060:          strcpy(filename, filename+1);              /* remove left '       */
1061:          filename[strlen(filename)-1] = image_type; /* replace right '     */
1062:                                                     /* with image suffix   */
1063:          num_components = create_pairs(filename);   /* num_... holds error code */
1064:          if (num_components == 1)                   /* fail immediately */
1065:             break;
1066:          else
1067:             if (num_components == 2) {              /* try again       */
1068:                 textcolor(ERROR_COLOR);
1069:                 cprintf("Terminate making pairs?\n\r");
1070:                 if (toupper(confirm())=='Y')
1071:                     break;
1072:             }
1073:          fscanf(file_ptr, "%*d%d", &num_components);
1074:          for (num_components=num_components+3; num_components>0; num_components--)
1075:             fgets(filename, 100, file_ptr);
1076:          }
1077:
1078:         /*** remove lock file ***/
1079:     if (unlink(LOCK_FILE)!=0)
1080:         print_error("Unable to remove lock file.  Check disk.");
1081:     }   /* make_image_pairs */
1082: /*-------------------------------------------------------------------------*/
1083:
1084:
1085:
1086: /*-------------------------------------------------------------------------*/
1087: byte make_images(void)
1088:
1089:     /*      This module gets the name of a data file containing
1090:         component information, determines if the user wishes to generate
1091:         rectangular or gaussian images, whether the images are to be
1092:         summed or if the images are individual, and whether the images are
1093:         to be created as pairs or not.  Information in the data file is
1094:         copied to the file CREATE.DAT where it will be read by the create
1095:         programs.  The create programs are DOS executables which are run
1096:         by exiting this program with an errorlevel corresponding to
1097:         which program should be run.  When control is returned to this
1098:         program it calls make_image_pairs to create pairs if they were
1099:         to be generated.  This is done by setting a semaphore in the
```

```
1100:         current directory (LOCK.TMP) which indicates that control is to
1102:         be returned to the texture generation menu.                    */
1103:
1104:     {
1105:     char    command[100];
1106:     char    filename[80];
1107:     FILE    *file_ptr;
1108:     char    image_type;
1109:     char    window_type;
1110:
1111:
1112:         /*** setup screen ***/
1113:     print_title("GEXPT 2: Make Images\n\n");
1114:
1115:         /*** determine component data filename ***/
1116:     get_input("Enter name of file containing component data (no extension):", "%s", filename);
1117:     if (filename[0]==EXPAND_CHAR) {
1118:         strcpy(command, DATA_DIR);
1119:         strcat(command, filename+1);
1120:         strcpy(filename, command);
1121:         }
1122:     textcolor(MENU_COLOR);
1123:     sprintf(command, "copy %s.DAT " CREATE_FILE, filename);
1124:     system(command);
1125:     cprintf("\n\r");
1126:
1127:         /*** create a lock file which indicates if the pairs are to be ***
1128:          *** created or not                                            ***/
1129:     get_input("Do you wish to make pairs <N>:", "%c", &image_type);
1130:     if (image_type==' ')
1131:         image_type = 'n';
1132:     if ((file_ptr=fopen(LOCK_FILE, "wt"))==NULL) {
1133:         print_error("Unable to create lock file.  Check disk.");
1134:         return(REDISPLAY);
1135:         }
1136:     fprintf(file_ptr, "%c", image_type);
1137:
1138:          /*** determine if single or summed images will be generated ***
1139:           *** and save info in lock file                               ***/
1140:     for (image_type=' '; toupper(image_type)!='S' && toupper(image_type)!='I'; )
1141:         get_input("Enter 's' for summed images or 'i' for individual images:", "%c", &image_type);
1142:     fprintf(file_ptr, "%c", image_type);
1143:
1144:         /*** determine if gaussian or rectangular images will be generated ***
1145:          *** and save in lock file                                        ***/
1146:     for (window_type=' '; toupper(window_type)!='G' && toupper(window_type)!='R'; )
1147:         get_input("Enter 'g' for Gaussian window or 'r' for rectangular window:", " %c", &window_type);
1148:     fprintf(file_ptr, "%c", window_type);
1149:
1150:         /*** close lock file ***/
1151:     if (fclose(file_ptr)) {
1152:         print_error("Unable to close lock file.  Check disk.");
1153:         return(REDISPLAY);
1154:         }
```

```
1155:
1157:         /*** set the return errorcode to a value which indicates which ***
1158:          *** create program is to be run                                ***/
1159:     if (toupper(window_type) == 'G')
1160:         if (toupper(image_type) == 'I') {
1161:             clrscr();
1162:             cprintf("%*cNow creating individual images with Gaussian window:\n\r", ENTRY_INDENT, ' ');
1163:             sleep(1.0);
1164:             return(201);
1165:             }
1166:         else {
1167:             clrscr();
1168:             cprintf("%*cNow creating summed images with Gaussian window:\n\r", ENTRY_INDENT, ' ');
1169:             sleep(1.0);
1170:             return(202);
1171:             }
1172:     else
1173:         if (toupper(image_type) == 'I') {
1174:             clrscr();
1175:             cprintf("%*cNow creating individual images with Rectangular window:\n\r", ENTRY_INDENT, ' ');
1176:             sleep(1.0);
1177:             return(203);
1178:             }
1179:         else {
1180:             clrscr();
1181:             cprintf("%*cNow creating summed images with Rectangular window:\n\r", ENTRY_INDENT, ' ');
1182:             sleep(1.0);
1183:             return(204);
1184:             }
1185:
1186:     }   /* make_images */
1187: /*-------------------------------------------------------------------------*/
```

```
/*****************************************************************************

            FILENAME:           gexpt2b.h
            PROGRAMMER:         Christopher Voltz - UDRI
            CREATED:            -1/8810.18
            LAST MODIFIED:      -1/8903.27
            INTERFACE PROTOCOL: TURBO C 2.0
            USAGE:              #include <gexpt2b.h>


        This module contains the headers for the following routine(s):

    display_keyboard_data_collection_menu   -> This routine is responsible
        for displaying the constant stimuli menu.  It displays the options
        available, records a keystroke, and executes the appropriate
        function.  It will return a value indicating whether the menu
        should redisplayed (called again) or not.

    flash_screens   -> This routine displays each screen for a specified
        duration and then proceeds to the next screen in the sequence.  A
        sample presentation might be: noise, adapting, noise, stimulus, mask,
        and noise.

*****************************************************************************/



/**********************
 * FUNCTION PROTOTYPES *
 **********************/

byte display_keyboard_data_collection_menu (void);
void flash_screens (int adapt_duration, int duration);
```

```
/*****************************************************************************

            FILENAME:               gexpt2b.c
            PROGRAMMER:             Christopher Voltz - UDRI
            CREATED:                -1/8810.18
            LAST MODIFIED:          -1/8904.21
            INTERFACE PROTOCOL:     TURBO C 2.0
            USAGE:                  #include <gexpt2b.h>


        This module contains code for the following routines:

    analyze_data    -> This routine analyzes the constant stimuli data and
        writes a summary to the stream passed to it.  If the stream is a NULL
        pointer, then the user is prompted for a stream to write the summary
        data to.

    collect_data    -> This routine presents stimuli, collects the data, and
        produces .RAW and .SUM files.  The stimuli are presented using the
        constant stimuli method.  The keyboard is used for data input.

    display_group_sequence  -> This routine displays the groups in a
        subject's group sequence file.

    display_keyboard_data_collection_menu   -> This routine allows the user
        to choose, from a menu, whether he would like to: set the group
        presentation sequence, or test the response box, or collect data, or
        analyze data, or graph a summary file.

    flash_screens   -> This routine displays each screen for a specified
        duration and then proceeds to the next screen in the sequence.  A
        sample presentation might be: noise, adapting, noise, stimulus, mask,
        and noise.

    graph_summary_file  -> This routine graphs the summary file created by
        the analyze_data routine.

    randomize_trials    -> This routine creates the file which the
        collect_data routine reads.  The file consists of the filenames of
        images to be displayed and their relevant data, ie. phase,
        orientation, frequency, etc.

    set_group_sequence ->   This routine determines the selection and order
        the groups are presented in.

*****************************************************************************/



/****************
 * HEADER FILES *
 ****************/

    /*** TURBO C header files ***/
```

```
#include <alloc.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

    /*** program specific header files ***/
#include <constant.h>           /* program constants to define system   */
                                /* parameters for included files        */
#if DT2871
#    include <dt2871.h>        /* routines to control DT-2871 board    */
#else
#    include <pcwrap.h>        /* wrapper for pcvision routines        */
#endif
#include <response.h>           /* routines to control response box     */
#include <toolbox.h>            /* general utility routines             */
#include <gexpt2.h>             /* header file of main program          */
#include <gexpt2b.h>            /* this module's header file            */


    /************************
     *  FUNCTION PROTOTYPES  *
     ************************/

static void analyze_data(void);
static void collect_data(void);
static void display_group_sequence(void);
static void graph_summary_file(void);
static byte randomize_trials(char *filename);
static void set_group_sequence(void);


/************************
 * FUNCTION DEFINITIONS *
 ************************/

/*--------------------------------------------------------------------------*/
static void analyze_data(void)
    {
    struct node {
        int         level;      /* bandwidth level, if used             */
        int         freq;       /* number of frequencies in image       */
        int         orient;     /* number of orientations in image      */
        int         count;      /* num of x => n                        */
        int         sum;        /* sum of x                             */
        int         sum_2;      /* sum of x^2                           */
        int         ss_sum;     /* sum of x for same standard           */
```

```
110:        int         ss_sum_2;   /* sum of x^2 for same standard         */
112:        int         ss_count;   /* num of x for same standard => s_n    */
113:        int         ds_sum;     /* sum of x for different standard      */
114:        int         ds_sum_2;   /* sum of x^2 for different standard    */
115:        int         ds_count;   /* num of x for different standard      */
116:        struct node *next;      /* link to next node in list            */
117:        );
118:
119:    boolean     bandwidth;          /* indicates if R (bandwidth) analysis  */
120:    int         band_level_c=0;     /* bandwidth level of comparison        */
121:    int         band_level_s=0;     /* bandwidth level of standard          */
122:    char        filename[60];       /* filename of file to open             */
123:    FILE        *file_in;           /* pointer to input file                */
124:    FILE        *file_out;          /* pointer to output file               */
125:    boolean     found;              /* indicates if node currently exists   */
126:    int         num_freq_c;         /* number of frequencies in comparison  */
127:    int         num_freq_s;         /* number of frequencies in standard    */
128:    int         num_orient_c;       /* number of orientations in comparison */
129:    int         num_orient_s;       /* number of orientations in standard   */
130:    struct node *occur;             /* pointer to lists of nodes            */
131:    struct node *ptr;               /* pointer in lists of nodes            */
132:    int         response;           /* response by subject                  */
133:    int         same;               /* if left and right images are same    */
134:    int         side;               /* side standard was presented on       */
135:    char        string[100];        /* temporary string                     */
136:    struct node *temp_ptr;          /* temporary pointer                    */
137:
138:
139:            /*** print title screen ***/
140:    print_title("GEXPT 2: Analyze Data\n\n");
141:
142:        /*** determine filename of output file ***/
143:    get_input("Enter filename of output file (.SUM):", "%s", string);
144:    if (string[0]==EXPAND_CHAR)
145:        sprintf(filename, "%s%s.SUM", DATA_DIR, string+1);
146:    else
147:        sprintf(filename, "%s.SUM", string);
148:    if ((file_out=fopen(filename, "wt"))==NULL) {
149:        print_error("Unable to open output file");
150:        return;
151:        }   /* if */
152:    fprintf(file_out, "%s consists of:\n", filename);
153:
154:        /*** determine if R (bandwidth) analysis is to be done ***/
155:    get_input("Enter Y if bandwidth analysis to be done:", "%c", &string[0]);
156:    bandwidth = (toupper(string[0])=='Y');
157:
158:        /*** determine if the files to be analyzed are old files ***/
159:    textcolor(MENU_COLOR);
160:    cprintf("%*cHit CR to end filename entry.\n\r\n", ENTRY_INDENT, ' ');
161:    occur = NULL;
162:
163:            /*** for each group, analyze file ***/
164:    for (strcpy(filename, "CV"); strlen(filename)!=0; ) {
```

```
165:
167:            /*** determine filename and open file ***/
168:        get_input("Enter name of raw data file to analyze (.RAW):", "", filename);
169:        if (strlen(filename)==0)
170:            continue;
171:        if (filename[0]==EXPAND_CHAR)
172:            sprintf(string, "%s%s.RAW", DATA_DIR, filename+1);
173:        else
174:            sprintf(string, "%s.RAW", filename);
175:        if ((file_in=fopen(string, "rt"))==NULL) {
176:            print_error("Unable to find specified file.");
177:            continue;
178:            }
179:        fprintf(file_out, "%*c%s\n", ENTRY_INDENT, ' ', string);
180:
181:            /*** read in data and append to current list ***/
182:        while (!feof(file_in)) {
183:            fgets(string, 100, file_in);
184:            if (bandwidth)
185:                sscanf(string, "%*d %*d %d %d %d %d %d %*d %d %d %*d %d %*d %*d %*d %d %d",
186:                       &num_orient_s, &num_freq_s, &num_orient_c, &num_freq_c,
187:                       &side, &response, &same, &band_level_s, &band_level_c);
188:            else
189:                sscanf(string, "%*d %*d %d %d %d %d %*d %*d %d %d %*d %d %*d %*d %*d %*d",
190:                       &num_orient_s, &num_freq_s, &num_orient_c, &num_freq_c,
191:                       &response, &same);
192:            for (ptr=occur, found=FALSE; ptr!=NULL && !found;) {
193:                found = ptr->freq==num_freq_c && ptr->orient==num_orient_c &&
194:                        ptr->level==band_level_c;
195:                if (!found)
196:                    ptr = ptr->next;
197:                }
198:            if (!found) {
199:                if ((ptr=(struct node *)malloc(sizeof(struct node)))==NULL) {
200:                    print_error("Insufficient memory.");
201:                    for (ptr=occur; ptr!=NULL; ptr=occur) {
202:                        occur = occur->next;
203:                        free(ptr);
204:                        }
205:                    fclose(file_in);
206:                    fclose(file_out);
207:                    return;
208:                    }   /* if */
209:                ptr->next = occur;
210:                ptr->level = band_level_c;
211:                ptr->freq = num_freq_c;
212:                ptr->orient = num_orient_c;
213:                ptr->sum = ptr->sum_2 = ptr->count = 0;
214:                ptr->ss_sum = ptr->ss_sum_2 = ptr->ss_count = 0;
215:                ptr->ds_sum = ptr->ds_sum_2 = ptr->ds_count = 0;
216:                occur = ptr;
217:                }   /* if */
218:            ptr->sum += response;
219:            ptr->sum_2 += pow(response,2);
```

```
220:            ptr->count += 1;
222:            if (num_freq_s==num_freq_c && num_orient_s==num_orient_c &&
223:                band_level_s==band_level_c)
224:                if (same) {
225:                    ptr->ss_sum += response;
226:                    ptr->ss_sum_2 += pow(response,2);
227:                    ptr->ss_count += 1;
228:                    }
229:                else {
230:                    ptr->ds_sum += response;
231:                    ptr->ds_sum_2 += pow(response,2);
232:                    ptr->ds_count += 1;
233:                    }
234:            }  /* while */
235:        fclose(file_in);
236:
237:        }  /* for */
238:
239:        /*** sort nodes by orientation, frequency, and bandwidth level ***/
240:    for (ptr=occur; ptr!=NULL; ptr=ptr->next)
241:        for (temp_ptr=ptr; temp_ptr!=NULL; temp_ptr=temp_ptr->next)
242:            if ( (ptr->freq<temp_ptr->freq) ||
243:                 ((ptr->freq==temp_ptr->freq) && (ptr->orient<temp_ptr->orient)) ||
244:                 ((ptr->freq==temp_ptr->freq) && (ptr->orient==temp_ptr->orient) &&
245:                  (ptr->level>temp_ptr->level)) ) {
246:                swap_int(&(ptr->level),    &(temp_ptr->level));
247:                swap_int(&(ptr->orient),   &(temp_ptr->orient));
248:                swap_int(&(ptr->freq),     &(temp_ptr->freq));
249:                swap_int(&(ptr->count),    &(temp_ptr->count));
250:                swap_int(&(ptr->sum),      &(temp_ptr->sum));
251:                swap_int(&(ptr->sum_2),    &(temp_ptr->sum_2));
252:                swap_int(&(ptr->ss_sum),   &(temp_ptr->ss_sum));
253:                swap_int(&(ptr->ss_sum_2), &(temp_ptr->ss_sum_2));
254:                swap_int(&(ptr->ss_count), &(temp_ptr->ss_count));
255:                swap_int(&(ptr->ds_sum),   &(temp_ptr->ds_sum));
256:                swap_int(&(ptr->ds_sum_2), &(temp_ptr->ds_sum_2));
257:                swap_int(&(ptr->ds_count), &(temp_ptr->ds_count));
258:                }
259:
260:        /*** write mean and standard error and close file ***/
261:    if (bandwidth)
262:        fprintf(file_out, "\n\n%.2s  %.2s  %.2s  %.9s  %.7s  %.10s  %.9s  %.10s  %.9s\n",
263:                "SF", "OR", "BW", "   MEAN  ", "  SEM  ", "SAMES MEAN",
264:                "SAMES SEM", "DIFFS MEAN", "DIFFS SEM");
265:    else
266:        fprintf(file_out, "\n\n%.2s  %.2s  %.9s  %.7s  %.10s  %.9s  %.10s  %.9s\n",
267:                "SF", "OR", "   MEAN  ", "  SEM  ", "SAMES MEAN",
268:                "SAMES SEM", "DIFFS MEAN", "DIFFS SEM");
269:    for (ptr=occur; ptr!=NULL; ptr=ptr->next) {
270:        if (ptr->count>1)
271:            if (bandwidth)
272:                fprintf(file_out, "%2d  %2d  %2d  %9.5f  %7.5f", ptr->freq,
273:                        ptr->orient, ptr->level, (float)((float)ptr->sum/ptr->count),
274:                        (float)sqrt((ptr->sum_2 - pow(ptr->sum, 2) / ptr->count) /

FILE=GEXPT2B.C    Fri Jun 16 01:58:16 1989    PAGE=5
```

```
                        (ptr->count)) / sqrt(ptr->count));
            else
                fprintf(file_out, "%2d  %2d  %9.5f  %7.5f", ptr->freq,
                        ptr->orient, (float)((float)ptr->sum/ptr->count),
                        (float)sqrt((ptr->sum_2 - pow(ptr->sum, 2) / ptr->count) /
                        (ptr->count)) / sqrt(ptr->count));
        else
            fprintf(file_out, "%2d  %2d    -----      -----  ", ptr->freq,
                    ptr->orient);
        if (ptr->ss_count>1)
            fprintf(file_out, "  %10.5f  %9.5f", (float)((float)ptr->ss_sum /
                    ptr->ss_count), (float)sqrt((ptr->ss_sum_2 -
                    pow(ptr->ss_sum, 2) / ptr->ss_count) / (ptr->ss_count)) /
                    sqrt(ptr->ss_count));
        else
            fprintf(file_out, "  %10c  %9c", ' ', ' ');
        if (ptr->ds_count>1)
            fprintf(file_out, "  %10.5f  %9.5f\n", (float)((float)ptr->ds_sum /
                    ptr->ds_count), (float)sqrt((ptr->ds_sum_2 -
                    pow(ptr->ds_sum, 2) / ptr->ds_count) / (ptr->ds_count)) /
                    sqrt(ptr->ds_count));
        else
            fprintf(file_out, "  %10c  %9c\n", ' ', ' ');
        }   /* for */

    fclose(file_out);

    }   /* analyze_data */
/*-------------------------------------------------------------------------*/



/*-------------------------------------------------------------------------*/
static void collect_data(void)

    {
    struct node {
        int         level;          /* bandwidth level, if used             */
        int         freq;           /* number of frequencies in image       */
        int         orient;         /* number of orientations in image      */
        int         count;          /* num of x => n                        */
        int         sum;            /* sum of x                             */
        int         sum_2;          /* sum of x^2                           */
        int         ss_sum;         /* sum of x for same standard           */
        int         ss_sum_2;       /* sum of x^2 for same standard         */
        int         ss_count;       /* num of x for same standard => s_n    */
        int         ds_sum;         /* sum of x for different standard      */
        int         ds_sum_2;       /* sum of x^2 for different standard    */
        int         ds_count;       /* num of x for different standard      */
        struct node *next;          /* pointer to next node in list         */
        };

#if ADAPT
    int         adapt_duration; /* number of fields to display adapting */
```

```
330: #endif
332:     boolean     bandwidth=FALSE;/* indicates if bandwidth stimuli in use    */
333:     struct date date_now;       /* current date                              */
334:     int         duration;       /* number of fields to display images        */
335:     int         eccentricity;   /* how far out center of image is            */
336:     char        filename[60];   /* filename of output file                   */
337:     FILE        *file_in;       /* input file pointer                        */
338:     FILE        *file_ptr;      /* general file pointer                      */
339:     header_type header;         /* header for image files                    */
340:     int         index;          /* general loop control variable             */
341:     int         index_2;        /* second general loop control variable      */
342:     int         level_1=0;      /* bandwidth level in image 1                */
343:     int         level_2=0;      /* bandwidth level in image 2                */
344:     int         num_freq_1;     /* number of frequencies in image 1          */
345:     int         num_freq_2;     /* number of frequencies in image 2          */
346:     int         num_groups;     /* number of groups to run in session        */
347:     int         num_orient_1;   /* number of orientations in image 1         */
348:     int         num_orient_2;   /* number of orientations in image 2         */
349:     struct node *occur;         /* pointer to list of nodes                  */
350:     int         phase_1;        /* phase number of image 1                   */
351:     int         phase_2;        /* phase number of image 2                   */
352:     struct node *ptr;           /* general node pointer                      */
353:     FILE        *raw_file;      /* raw data file pointer                     */
354:     int         response;       /* subject's response                        */
355:     int         session;        /* session number                            */
356:     int         side;           /* side that signal is on                    */
357:     char        string[100];    /* temp variable                             */
358:     char        string_2[100];  /* temp variable                             */
359:     int         subject_num;    /* subject number                            */
360:     FILE        *sum_file;      /* summary file pointer                      */
361:     struct node *temp_ptr;      /* additional general node pointer           */
362:     struct time time_now;       /* current time                              */
363:     int         val;            /* temp variable                             */
364:
365:
366:             /*** setup screen ***/
367:     print_title("GEXPT 2: Subject Data Entry\n\n");
368:
369:             /*** initialize graphics hardware ***/
370:     initialize_hardware();
371:
372:             /*** get data to create filename ***/
373:     get_input("Subject number:", "%d", &subject_num);
374:     get_input("Session number:", "%d", &session);
375:     get_input("Eccentricity (1=0.75 deg, 2=20 deg):", "%d", &eccentricity);
376:     get_input("Stimulus duration (0=167 ms, 1=334 ms):", "%d", &duration);
377:     if (duration)
378:         duration = 20;  /* 20 fields => 10 frames => 334 ms */
379:     else
380:         duration = 10;  /* 10 fields => 5 frames => 167 ms  */
381:     get_input("Number of groups to run (0 to abort):", "%d", &num_groups);
382:     if (!num_groups)
383:         return;
384:
```

```
385:            /*** open data files ***/
387:    sprintf(filename, "%s%04.4d%04.4d.SUM", DATA_DIR, subject_num, session);
388:    if ((sum_file=fopen(filename, "wt"))==NULL) {
389:        print_system_error();
390:        return;
391:        }
392:    fprintf(sum_file, "GEXPT 2 Summary file: %s\t", filename);
393:
394:    sprintf(filename, "%s%04.4d%04.4d.RAW", DATA_DIR, subject_num, session);
395:    if ((raw_file=fopen(filename, "wt"))==NULL) {
396:        print_system_error();
397:        return;
398:        }
399:
400:            /*** save header in data file ***/
401:    getdate(&date_now);
402:    gettime(&time_now);
403:    fprintf(sum_file, "CREATED: %d/%d/%d at %02d:%02d:%02d\n",
404:            date_now.da_mon, date_now.da_day, date_now.da_year,
405:            time_now.ti_hour, time_now.ti_min, time_now.ti_sec);
406:
407:            /*** get noise screen filename ***/
408:    filename[0] = 0;
409:    get_input("Enter noise screen filename (``###.IMG  CR for none):", "%s", filename);
410:    if (filename[0]!=0) {
411:        strcpy(string, IMAGE_DIR);
412:        strcat(string, "G");
413:        strcat(string, filename);
414:        strcpy(filename, string);
415:        fprintf(sum_file, "Noise screen: %s\n", filename);
416:        strcat(filename, "GL.IMG");
417:        read_image(NOISE_BUFFER, LEFT, filename, header);
418:        strcat(string, "GR.IMG");
419:        read_image(NOISE_BUFFER, RIGHT, string, header);
420:        }
421:
422:            /*** get adapting screen filename ***/
423:    filename[0] = 0;
424:    get_input("Enter adapting screen filename (``###.IMG  CR for none):", "%s", filename);
425:    if (filename[0]==0)
426:        adapt_flag = FALSE;
427:    else {
428:        adapt_flag = TRUE;
429:        strcpy(string, IMAGE_DIR);
430:        strcat(string, "G");
431:        strcat(string, filename);
432:        strcpy(filename, string);
433:        fprintf(sum_file, "Adapting screen filename: %s\n", filename);
434:        strcat(filename, "GL.IMG");
435:        read_image(ADAPT_BUFFER, LEFT, filename, header);
436:        strcat(string, "GR.IMG");
437:        read_image(ADAPT_BUFFER, RIGHT, string, header);
438: #if ADAPT
439:        get_input("Adapting duration (0=167 ms, 1=334 ms):", "%d", &adapt_duration);
```

```
440:         if (adapt_duration)
442:             adapt_duration = 20;     /* 20 fields => 10 frames => 334 ms */
443:         else
444:             adapt_duration = 10;     /* 10 fields => 5 frames => 167 ms  */
445: #endif
446:         )
447:
448:             /*** initialize node pointer ***/
449:     occur = NULL;
450:
451:             /*** erase old randomized trial data ***/
452:     sprintf(string_2, "%s%s", EXE_DIR, TEMP_FILE);
453:     unlink(string_2);
454:
455:             /*** for each group repeat the following ***/
456:     while (num_groups--) {
457:
458:             /*** open group sequence file and read current group;   ***
459:              *** also update current group pointer                 ***/
460:         sprintf(string_2, "%ssub1%04.4d.dat", EXE_DIR, subject_num);
461:         if ((file_in=fopen(string_2, "rt"))==NULL) {
462:             print_system_error();
463:             fclose(raw_file);
464:             fclose(sum_file);
465:             return;
466:             }
467:         sprintf(string, "%s%s", EXE_DIR, TEMP_FILE_2);
468:         if ((file_ptr=fopen(string, "wt"))==NULL) {
469:             print_system_error();
470:             fclose(raw_file);
471:             fclose(sum_file);
472:             return;
473:             }
474:         fscanf(file_in, "%d %d\n", &val, &index_2);
475:         fprintf(file_ptr, "%d %d\n", val, (++index_2 > val ? 1 : index_2));
476:         for (index=1; index<index_2 && !feof(file_in); index++) {
477:             fgets(filename, 60, file_in);
478:             fprintf(file_ptr, "%s", filename);
479:             }
480:         for (; index<=val && !feof(file_in); index++) {
481:             fgets(string, 100, file_in);
482:             fprintf(file_ptr, "%s", string);
483:             }
484:         fclose(file_ptr);
485:         fclose(file_in);
486:         unlink(string_2);
487:         sprintf(string, "%s%s", EXE_DIR, TEMP_FILE_2);
488:         rename(string, string_2);
489:
490:
491:             /*** open group directory and determine set filename ***/
492:         sprintf(string, "%s%s", EXE_DIR, GROUP_DIR);
493:         if ((file_in=fopen(string, "rt"))==NULL) {
494:             print_system_error();
```

```
495:            fclose(raw_file);
497:            fclose(sum_file);
498:            return;
499:            }
500:        fscanf(file_in, "%*d\n");   /* skip highest group number */
501:        for (strcpy(string, ""); strcmp(string, filename) && !feof(file_in); ) {
502:            fgets(string, 100, file_in);
503:            fgets(string_2, 100, file_in);
504:            }
505:        if (strcmp(string, filename)) {
506:            print_error("Could not find specified group.");
507:            fclose(raw_file);
508:            fclose(sum_file);
509:            return;
510:            }
511:        fclose(file_in);
512:
513:            /*** create randomized list of trial info ***/
514:        sprintf(string, "%s%.*s", EXE_DIR, strlen(string_2)-1, string_2);
515:        strcpy(string_2, string);
516:        if (!bandwidth)
517:            bandwidth = toupper(string_2[strlen(string_2)-5]) == 'B';
518:        else
519:            if (toupper(string_2[strlen(string_2)-4]) == 'B') {
520:                print_error("All groups must be either bandwidth or not bandwidth.");
521:                fclose(raw_file);
522:                fclose(sum_file);
523:                return;
524:                }
525:        if (randomize_trials(string_2)) {
526:            fclose(raw_file);
527:            fclose(sum_file);
528:            return;
529:            }
530:
531:    }   /* while */
532:
533:        /*** wait for subject to signal ready ***/
534:    textcolor(MENU_COLOR);
535:    cprintf("\n\r\n%*cPress any key to begin.\n\r\n",
536:            ENTRY_INDENT, ' ', ENTRY_INDENT, ' ');
537:    getch();
538:
539:        /*** display sequence of trials; get and record responses ***/
540:    sprintf(string, "%s%s", EXE_DIR, TEMP_FILE);
541:    if ((file_in=fopen(string, "rt"))==NULL) {
542:        print_error("Unable to open temp file.");
543:        fclose(raw_file);
544:        fclose(sum_file);
545:        return;
546:        }
547:    while (!feof(file_in)) {
548:        fscanf(file_in, "%s %d %d %d %s %d %d %d %d", string,
549:              &num_orient_1, &num_freq_1, &phase_1, string_2, &num_orient_2,
```

```
550:                &num_freq_2, &phase_2, &side);
552:        if (bandwidth)
553:            fscanf(file_in, "%d %d\n", &level_1, &level_2);
554:        else
555:            fscanf(file_in, "\n");
556:        if (bandwidth)
557:            cprintf("\n\r%*c'%s' %2d %2d %2d %1d  '%s' %2d %2d %2d %1d   %5s\n\r",
558:                    ENTRY_INDENT, ' ', string, num_freq_1, num_orient_1,
559:                    phase_1, level_1, string_2, num_freq_2, num_orient_2,
560:                    phase_2, level_2, (side?"Left ":"Right"));
561:        else
562:            cprintf("\n\r%*c'%s' %2d %2d %2d  '%s' %2d %2d %2d   %5s\n\r",
563:                    ENTRY_INDENT, ' ', string, num_freq_1, num_orient_1,
564:                    phase_1, string_2, num_freq_2, num_orient_2, phase_2,
565:                    (side?"Left ":"Right"));
566:        if (side==RIGHT) {
567:            swap_int(&level_1, &level_2);
568:            swap_int(&num_orient_1, &num_orient_2);
569:            swap_int(&num_freq_1, &num_freq_2);
570:            swap_int(&phase_1, &phase_2);
571:            }
572:        display_buffer(ADAPT_BUFFER);
573:        read_2_images(SIGNAL_BUFFER, string, header, LEFT, string_2, header);
574:        flash_screens(0 /* adapt_duration */, duration);
575:        for (ptr=occur, index_2=FALSE; ptr!=NULL && !index_2; ) {
576:            index_2 = (ptr->freq==num_freq_2 && ptr->orient==num_orient_2 && ptr->level==level_2);
577:            if (!index_2)
578:                ptr=ptr->next;
579:            }
580:        if (!index_2) {
581:            if ((ptr=malloc(sizeof(struct node)))==NULL) {
582:                print_error("Unable to add additional set.");
583:                for (ptr=occur; ptr!=NULL; ptr=occur->next, free(occur), occur=ptr);
584:                fclose(raw_file);
585:                fclose(sum_file);
586:                return;
587:                }
588:            ptr->freq = num_freq_2;
589:            ptr->orient = num_orient_2;
590:            ptr->level = level_2;
591:            ptr->sum = ptr->sum_2 = ptr->count = 0;
592:            ptr->ss_sum = ptr->ss_sum_2 = ptr->ss_count = 0;
593:            ptr->ds_sum = ptr->ds_sum_2 = ptr->ds_count = 0;
594:            ptr->next = occur;
595:            occur = ptr;
596:            }   /* if */
597:        for (index_2=FALSE; !index_2; ) {
598:            while (kbhit())         /* clear buffer */
599:                getch();
600:            cprintf("%*cEnter response: ", ENTRY_INDENT, ' ');
601:            textcolor(ENTRY_COLOR);
602:            response = getche();
603:            textcolor(MENU_COLOR);
604:            cprintf("\n\r");
```

```
605:            if (response==ESC_KEY) {
607:                for (ptr=occur; ptr!=NULL; ptr=occur->next, free(occur), occur=ptr);
608:                fclose(raw_file);
609:                fclose(sum_file);
610:                return;
611:                }
612:            index_2 = (response>'0' && response<'8');
613:            if (!index_2) {
614:                sound(BEEP_FREQUENCY);
615:                delay(BEEP_DELAY);
616:                nosound();
617:                }
618:            }   /* for */
619:        response -= '0';
620:        fprintf(raw_file, "%1d %1d  %2d %2d  %2d %2d  %1d %1d %1d %2d %1d "
621:                "%1d %1d %1d ", subject_num, session, num_orient_1,
622:                num_freq_1, num_orient_2, num_freq_2, side, eccentricity,
623:                response, num_freq_2*num_orient_2,
624:                (int)(strnicmp(string, string_2, strlen(string)-5)==0 ? 1 : 0),
625:                (int)(duration==10 ? 0 : 1), phase_1, phase_2);
626:        if (bandwidth)
627:            fprintf(raw_file, " %1d %1d\n", level_1, level_2);
628:        else
629:            fputc('\n', raw_file);
630:        ptr->sum += response;
631:        ptr->sum_2 += pow(response,2);
632:        ptr->count++;
633:        if (num_orient_1==num_orient_2 && num_freq_1==num_freq_2 && level_1==level_2)
634:            if (strnicmp(string, string_2, strlen(string)-5)==0) {
635:                ptr->ss_sum += response;
636:                ptr->ss_sum_2 += pow(response, 2);
637:                ptr->ss_count++;
638:                }
639:            else {
640:                ptr->ds_sum += response;
641:                ptr->ds_sum_2 += pow(response, 2);
642:                ptr->ds_count++;
643:                }
644:        }
645
646:        /*** inform user it is end of group and close raw file ***/
647:    sound(BEEP_FREQUENCY);
648:    delay(BEEP_DELAY/2);
649:    nosound();
650:    delay(BEEP_DELAY/2);
651:    sound(BEEP_FREQUENCY);
652:    delay(BEEP_DELAY/2);
653:    nosound();
654:    fclose(raw_file);
655:
656:        /*** sort nodes by orientation and frequency and bandwidth level ***/
657:    for (ptr=occur; ptr!=NULL; ptr=ptr->next)
658:        for (temp_ptr=ptr; temp_ptr!=NULL; temp_ptr=temp_ptr->next)
659:            if ( (ptr->freq<temp_ptr->freq) ||
```

```
660:                ((ptr->freq==temp_ptr->freq) && (ptr->orient<temp_ptr->orient)) ||
662:                ((ptr->freq==temp_ptr->freq) && (ptr->orient==temp_ptr->orient) &&
663:                 (ptr->level>temp_ptr->level)) ) {
664:              swap_int(&(ptr->level),    &(temp_ptr->level));
665:              swap_int(&(ptr->orient),   &(temp_ptr->orient));
666:              swap_int(&(ptr->freq),     &(temp_ptr->freq));
667:              swap_int(&(ptr->count),    &(temp_ptr->count));
668:              swap_int(&(ptr->sum),      &(temp_ptr->sum));
669:              swap_int(&(ptr->sum_2),    &(temp_ptr->sum_2));
670:              swap_int(&(ptr->ss_sum),   &(temp_ptr->ss_sum));
671:              swap_int(&(ptr->ss_sum_2), &(temp_ptr->ss_sum_2));
672:              swap_int(&(ptr->ss_count), &(temp_ptr->ss_count));
673:              swap_int(&(ptr->ds_sum),   &(temp_ptr->ds_sum));
674:              swap_int(&(ptr->ds_sum_2), &(temp_ptr->ds_sum_2));
675:              swap_int(&(ptr->ds_count), &(temp_ptr->ds_count));
676:              }
677:
678:          /*** write mean and standard error and close file ***/
679:      if (bandwidth)
680:          fprintf(sum_file, "\n\n%.2s  %.2s  %.2s  %.9s  %.7s  %.10s  %.9s  %.10s  %.9s\n",
681:                  "SF", "OR", "BW", "   MEAN  ", "  SEM  ", "SAMES MEAN",
682:                  "SAMES SEM", "DIFFS MEAN", "DIFFS SEM");
683:      else
684:          fprintf(sum_file, "\n\n%.2s  %.2s  %.9s  %.7s  %.10s  %.9s  %.10s  %.9s\n",
685:                  "SF", "OR", "   MEAN  ", "  SEM  ", "SAMES MEAN", "SAMES SEM",
686:                  "DIFFS MEAN", "DIFFS SEM");
687:      for (ptr=occur; ptr!=NULL; ptr=ptr->next) {
688:          if (ptr->count>1)
689:              if (bandwidth)
690:                  fprintf(sum_file, "%2d  %2d  %2d  %9.5f  %7.5f", ptr->freq,
691:                          ptr->orient, ptr->level, (float)((float)ptr->sum/ptr->count),
692:                          (float)sqrt((ptr->sum_2 - pow(ptr->sum, 2) / ptr->count) /
693:                          (ptr->count)) / sqrt(ptr->count));
694:              else
695:                  fprintf(sum_file, "%2d  %2d  %9.5f  %7.5f", ptr->freq,
696:                          ptr->orient, (float)((float)ptr->sum/ptr->count),
697:                          (float)sqrt((ptr->sum_2 - pow(ptr->sum, 2) / ptr->count) /
698:                          (ptr->count)) / sqrt(ptr->count));
699:          else
700:              fprintf(sum_file, "%2d  %2d    -----      -----  ", ptr->freq,
701:                      ptr->orient);
702:          if (ptr->ss_count>1)
703:              fprintf(sum_file, "  %10.5f  %9.5f", (float)((float)ptr->ss_sum /
704:                      ptr->ss_count), (float)sqrt((ptr->ss_sum_2 -
705:                      pow(ptr->ss_sum, 2) / ptr->ss_count) /
706:                      (ptr->ss_count))  / sqrt(ptr->ss_count));
707:          else
708:              fprintf(sum_file, "  %10c  %9c", ' ', ' ');
709:          if (ptr->ds_count>1)
710:              fprintf(sum_file, "  %10.5f  %9.5f\n", (float)((float)ptr->ds_sum /
711:                      ptr->ds_count), (float)sqrt((ptr->ds_sum_2 -
712:                      pow(ptr->ds_sum, 2) / ptr->ds_count) /
713:                      (ptr->ds_count)) / sqrt(ptr->ds_count));
714:          else

FILE=GEXPT2B.C    Fri Jun 16 01:58:16 1989    PAGE=13
```

```
715:             fprintf(sum_file, "  %10c  %9c\n", ' ', ' ');
717:         }   /* for */
718:
719:         /*** write ending comments and close file ***/
720:     while (kbhit())         /* clear buffer */
721:         getch();
722:     cprintf("\n\r\n");
723:     get_input("Enter comments (100 char max): ", "", string);
724:     fprintf(sum_file, "\n\nComments: %s\n", string);
725:     for (ptr=occur; ptr!=NULL; ptr=occur->next, free(occur), occur=ptr);
726:     fclose(sum_file);
727:
728:     }   /* collect_data */
729: /*------------------- ---------------------- ---------------------------------*/
730:
731:
732:
733: /*-------------------------------------------------------------------------*/
734: static void display_group_sequence(void)
735:
736:     /*      This module displays the information used by the randomize
737:         procedure.  Specifically, the group names are displayed.        */
738:
739:     {
740:     FILE    *file_in;                   /* input file                   */
741:     int     current_group;              /* current group number         */
742:     char    group_name[MAX_GROUP_CHAR]; /* name of group                */
743:     int     num_groups;                 /* number of groups in file     */
744:     int     subject_number;             /* subject's code number        */
745:     char    string[100];                /* temporary string             */
746:
747:
748:     print_title("GEXPT 2: Display Group Sequence\n\n");
749:
750:     get_input("Enter subject number:", "%d", &subject_number);
751:     sprintf(string, "%ssub1%4.4d.dat", EXE_DIR, subject_number);
752:     file_in = fopen(string, "rt");
753:     if (file_in==NULL)
754:         print_system_error();
755:     else {
756:         print_title("GEXPT 2: Display Group Sequence\n\n");
757:         fscanf(file_in, "%d %d\n", &num_groups, &current_group);
758:         cprintf("Subject %-d:\r\n", subject_number);
759:         cprintf("    has %-d groups.\r\n", num_groups);
760:         cprintf("    will use group number %-d next.\r\n", current_group);
761:         cprintf("\r\nThe groups are:\r\n");
762:         for (current_group=1; num_groups!=0; num_groups--, current_group++) {
763:             fgets(group_name, MAX_GROUP_CHAR, file_in);
764:             cprintf("%*c(%-d)  %s\r", ENTRY_INDENT, ' ', current_group,
765:                     group_name);
766:             }
767:         fclose(file_in);
768:         cprintf("\r\n\nPress any key to continue.");
769:         getch();
```

```
770:         }
772:
773:     }   /* display_group_sequence */
774: /*------------------------------------------------ ----------------------*/
775:
776:
777:
778: /*-------------------------------------------------------------------------*/
779: byte display_keyboard_data_collection_menu(void)
780:
781:     /*
782:         This module displays the keyboard data collection menu, gets a
783:         keystroke, executes the appropriate function, and returns either a non-zero
784:         value if the menu should be displayed again, or a zero value if the
785:         user selected the exit option.
786:     */
787:
788: {
789:
790:         /* setup the screen */
791:     print_title("GEXPT 2: Similarity Data Collect/Analyze Menu\n\n");
792:
793:         /* print options */
794:     print_option("A|Analyze data file");
795:     print_option("C|Collect data");
796:     print_option("D|Display group presentation sequence");
797:     print_option("G|Graph summary file");
798:     print_option("S|Set group presentation sequence");
799:     printf("\n");
800:     print_option("x|Exit menu");
801:     print_option("<ESC>|<ESC> Exit menu");
802:     cprintf("\r\n\n%*cEnter Option: ", ENTRY_INDENT, ' ');
803:
804:         /* get the user's option */
805:     textcolor(ENTRY_COLOR);
806:     switch (toupper(getche()))
807:     {
808:         case 'A':
809:             analyze_data();
810:             break;
811:         case 'C':
812:             collect_data();
813:             break;
814:         case 'D':
815:             display_group_sequence();
816:             break;
817:         case 'G':
818:             graph_summary_file();
819:             break;
820:         case 'S':
821:             set_group_sequence();
822:             break;
823:         case ESC_KEY:
824:             cprintf("\bX");
```

```
825:         case'X':
827:             return(EXIT_MENU);  /* exit this menu */
829:     }
829:
830:     return(REDISPLAY);  /* redisplay menu */
831:
832: }   /* display_keyboard_data_collection_menu */
833: /*-------------------------------------------------------------------------*/
834:
835:
836:
837: /*-------------------------------------------------------------------------*/
838: void flash_screens(int adapt_duration, int duration)
839:
840:     /*      This module flashes the stimulus screen on for a period of
841:         time, preceeded by either a noise screen, or an adapting screen
842:         followed the noise screen; and followed by the noise screen.       */
843:
844:     {
845:
846:     screen_hold(1);                         /* sync screen changes to   */
847:                                             /* vertical interrupt       */
848: #if ADAPT
849:     if (adapt_flag) {
850:         display_buffer(ADAPT_BUFFER);       /* show adapting harmonic   */
851:         screen_hold(adapt_duration);        /*   for # fields           */
852:         }
853:     else
854:         display_buffer(NOISE_BUFFER);
855: #endif
856:     display_buffer(NOISE_BUFFER);
857:     screen_hold(2*0.25*1000/16.7);          /* hold for 500 ms      */
858:
859:     display_buffer(SIGNAL_BUFFER);          /* show stimulus        */
860:     screen_hold(duration);                  /*   for # fields       */
861:
862:     display_buffer(NOISE_BUFFER);
863:
864: #pragma warn -par
865:     }   /* flash_screens */
866: #pragma warn .par
867: /*-------------------------------------------------------------------------*/
868:
869:
870:
871: /*-------------------------------------------------------------------------*/
872: static void graph_summary_file(void)
873:
874:     /*      This module reads in the summary data from a specified file and
875:         plots it on the screen using different line types.  The screen can
876:         be printed if an EGA screen dump utility has been activated.       */
877:
878:     {
879:     boolean bandwidth;          /* indicates if bandwidth levels in use */
```

```
880:    FILE    *file_in;           /* summary file pointer             */
882:    char    filename[100];      /* filename of summary file         */
883:    int     frequency;          /* frequency of current line        */
884:    int     last_frequency;     /* last frequency used              */
885:    int     last_orientation;   /* last orientation used            */
886:    int     level;              /* bandwidth level of current line  */
887:    int     line_type;          /* indicates current line type      */
888:    float   mean;               /* mean of current line             */
889:    int     num_lines;          /* number of sets displayed         */
890:    int     orientation;        /* orientation of current line      */
891:    char    temp[100];          /* used to expand filename          */
892:    int     x, y;               /* current graphics position        */
894:
895:        /*** setup screen and get name of input file ***/
896:    print_title("GEXPT 2: Graph Summary Files\n\n");
897:    get_input("Enter name of file to graph (.SUM):", "%s", filename);
898:    if (filename[0]==EXPAND_CHAR) {
899:        strcpy(temp, DATA_DIR);
900:        strcat(temp, filename+1);
901:        strcpy(filename, temp);
902:        }
903:    strcat(filename, ".SUM");
904:
905:        /*** open file ***/
906:    if ((file_in=fopen(filename, "rt"))==NULL) {
907:        print_error("Could not open graph file.");
908:        return;
909:        }
910:
911:        /*** see if bandwidth info was included in file ***/
912:    get_input("Enter Y if file contains bandwidth info:", "%c", &temp[0]);
913:    bandwidth = (toupper(temp[0])=='Y');
914:
915:        /*** initialize graphics hardware ***/
916:    if (registerfarbgidriver(EGAVGA_driver_far) < 0) {
917:        print_error("Graphics driver could not be registered.");
918:        fclose(file_in);
919:        return;
920:        }
921:    orientation = EGA;
922:    frequency = EGAHI;
923:    initgraph(&orientation, &frequency, NULL);
924:
925:        /*** setup screen ***/
926:    cleardevice();
927:
928:        /*** draw graph axes ***/
929:    setcolor(WHITE);
930:    moveto(100, 15);
931:    lineto(100, 305);
932:    lineto(550, 305);
933:    lineto(550, 15);
934:    lineto(100, 15);
```

```
935:
937:         /*** print graph labels ***/
938:     for (frequency=6; frequency>0; frequency--) {
939:         itoa(frequency, temp, 10);
940:         outtextxy(70, 13+(7-frequency)*41.4, temp);
941:         }
942:     settextjustify(BOTTOM_TEXT, CENTER_TEXT);
943:     settextstyle(DEFAULT_FONT, VERT_DIR, 1);
944:     outtextxy(40, 145, "Rating");
945:
946:     settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
947:     settextjustify(CENTER_TEXT, TOP_TEXT);
948:     if (bandwidth)
949:         for (frequency=1; frequency<6; frequency++) {
950:             itoa(frequency, temp, 10);
951:             outtextxy(101+frequency*74.667, 315, temp);
952:             }
953:     else
954:         for (frequency=1; frequency<9; frequency++) {
955:             itoa(frequency*8, temp, 10);
956:             outtextxy(101+frequency*49.778, 315, temp);
957:             }
958:     if (bandwidth)
959:         outtextxy(320, 335, "Bandwidth Level");
960:     else
961:         outtextxy(320, 335, "Number of Components");
962:     outtextxy(320, 0, filename);
963:     settextjustify(RIGHT_TEXT, TOP_TEXT);
964:
965:         /*** read data and plot lines ***/
966:     frequency = orientation = last_frequency = last_orientation = level = num_lines = 1;
967:     line_type = -1;
968:     while (!feof(file_in)) {
969:         if (line_type == -1)        /* if first time through    */
970:             do                      /* skip header              */
971:                 fgets(temp, 100, file_in);
972:             while (sscanf(temp, "%d %d")!=2);
973:         else
974:             fgets(temp, 100, file_in);
975:         if (bandwidth)
976:             sscanf(temp, "%d %d %d %f", &frequency, &orientation, &level, &mean);
977:         else
978:             sscanf(temp, "%d %d %f", &frequency, &orientation, &mean);
979:         if (frequency!=last_frequency || (bandwidth ? orientation!=last_orientation : 0)) {
983:             switch (line_type) {
981:                 case  0: line_type = 1;
982:                          setcolor(LIGHTBLUE);
983:                          setfillstyle(SOLID_FILL, LIGHTBLUE);
984:                          break;
985:                 case  1: line_type = 3;
986:                          setcolor(LIGHTCYAN);
987:                          setfillstyle(SOLID_FILL, LIGHTCYAN);
988:                          break;
989:                 case -1:
```

```
990:                case  3: line_type = 0;
992:                         setcolor(LIGHTRED);
993:                         setfillstyle(SOLID_FILL, LIGHTRED);
994:                         break;
995:                }   /* switch */
996:            itoa(frequency, temp, 10);
997:            outtextxy(585, 25+num_lines*10, temp);
998:            if (bandwidth) {
999:                itoa(orientation, temp, 10);
1000:               outtextxy(610, 25+num_lines*10, temp);
1001:               }
1002:           setlinestyle(line_type, 0, THICK_WIDTH);
1003:           line(625, 28+num_lines*10, 639, 28+num_lines*10);
1004:           moveto(bandwidth ? (101+level*74.667) : (101+frequency*orientation/8.0*49.778),
1005:                15+(7.0-mean)*41.4);
1006:           num_lines++;
1007:           )
1008:       else
1009:           lineto(bandwidth ? (101+level*74.667) : (101+frequency*orientation/8.0*49.778),
1010:                15+(7.0-mean)*41.4);
1011:       x = getx();
1012:       y = gety();
1013:       fillellipse(bandwidth ? (101+level*74.667) : (101+frequency*orientation/8.0*49.778),
1014:                15+(7.0-mean)*41.4,
1015:                2, 2);
1016:       moveto(x, y);
1017:       last_frequency = frequency;
1018:       last_orientation = orientation;
1019:       }   /* while */
1020:   fclose(file_in);
1021:
1022:       /*** wait for user to press key ***/
1023:   getch();
1024:
1025:       /*** shut down graphics system and restore CRT mode ***/
1026:   closegraph();
1027:
1028:   }   /* graph_summary_file */
1029: /*-------------------------------------------------------------------------*/
1030:
1031:
1032:
1033: /*-------------------------------------------------------------------------*/
1034: static byte randomize_trials(char *filename)
1035:
1036:   /*      This module reads in the the set info in the file given by
1037:       filename; adds to a file containing all the valid possible
1038:       combinations of set A with each of the other sets (on an image by
1039:       image basis); randomizes the order of the entries in the file; and
1040:       returns to the caller.                                          */
1041:
1042:   {
1043:   struct node_struct {
1044:           char                filename[60];
```

```
1045:           int                 freq;
1047:           int                 orient;
1048:           int                 phase;
1049:           int                 level;
1050:           struct node_struct  *next;
1051:           };
1052:   typedef struct node_struct node;
1053:   typedef char string[70];
1054:   struct list_struct {
1055:           string              s;
1056:           struct list_struct  *next;
1057:           };
1058:   typedef struct list_struct list;
1059:
1060:
1061:   int         bandwidth;              /* indicates bandwidth stimuli in use       */
1062:   FILE        *file_ptr;              /* file pointer                             */
1063:   int         index;                  /* general index variable                   */
1064:   node        *last;                  /* general pointer                          */
1065:   int         bandwidth_level;        /* level of bandwidth                       */
1066:   int         num;                    /* number of sets in use                    */
1067:   int         num_freq;               /* number of frequencies for a given set    */
1068:   int         num_orient;             /* number of orientations for a given set   */
1069:   list        *pairs;                 /* first entry in list of image pairs       */
1070:   list        *p1;                    /* general pointer                          */
1071:   list        *p2;                    /* general pointer                          */
1072:   node        *ptr;                   /* general pointer                          */
1073:   node        *set_ptr[MAX_SETS];     /* array of pointers to each set            */
1074:   int         size;                   /* size of array to be randomized           */
1075:   char        temp1[100];             /* temporary string variable                */
1076:
1077:
1078:       /*** determine if bandwidth stimuli in use ***/
1079:   bandwidth = (toupper(filename[strlen(filename)-5])=='B');
1080:
1081:       /*** open set file and read in set info ***/
1082:   if ((file_ptr=fopen(filename, "rt"))==NULL) {
1083:       print_system_error();
1084:       return (1);
1085:       }
1086:   fscanf(file_ptr, "%d ", &num);
1087:
1088:       /* allocate space for each head pointer and initialize each *
1089:        * next pointer                                             */
1090:   for (index=0; index<MAX_SETS; index++) {
1091:       if ((set_ptr[index]=(node *)malloc(sizeof(node)))==NULL) {
1092:           print_error("Insufficient memory.");
1093:           fclose(file_ptr);
1094:           for (; index>=0; index--)
1095:               free(set_ptr[index]);
1096:           return (1);
1097:           }  /* if */
1098:       (set_ptr[index])->next = NULL;
1099:       }  /* for */
```

```
1100:
1102:       /* read in the sets and add the set entry to the linked list of *
1103:        * entries                                                      */
1104:     for (index=size=0; index<num && !feof(file_ptr); index++) {
1105:         if (bandwidth)
1106:             fscanf(file_ptr, "%d %d %d\n", &num_orient, &num_freq, &bandwidth_level);
1107:         else
1108:             fscanf(file_ptr, "%d %d\n", &num_orient, &num_freq);
1109:         for (last=set_ptr[index], strcpy(temp1, ""); !feof(file_ptr) && strcmp(temp1, "."); last=ptr) {
1110:             if ((ptr=(node *)malloc(sizeof(node)))==NULL) {
1111:                 print_error("Insufficient memory to read in set info.");
1112:                 fclose(file_ptr);
1113:                 for (index=0; index<num; index++)
1114:                     for (last=ptr=set_ptr[index]; ptr!=NULL; ) {
1115:                         ptr = last->next;
1116:                         free(last);
1117:                         last = ptr;
1118:                         }  /* for */
1119:                 return (1);
1120:                 }  /* if */
1121:             fgets(temp1, 100, file_ptr);
1122:             temp1[strlen(temp1)-1] = '\0';  /* kill CR */
1123:             if (strcmp(temp1, ".")) {
1124:                 sscanf(temp1, "%s %d", ptr->filename, &(ptr->phase));
1125:                 ptr->freq = num_freq;
1126:                 ptr->orient = num_orient;
1127:                 if (bandwidth)
1128:                     ptr->level = bandwidth_level;
1129:                 else
1130:                     ptr->level = 0;
1131:                 ptr->next = last->next;
1132:                 last->next = ptr;
1133:                 size++;
1134:                 }
1135:             }  /* for */
1136:         }   /* for */
1137:         fclose(file_ptr);
1138:
1139:             /*** allocate space for array of image pair info ***/
1140:     pairs = malloc (sizeof(list));
1141:     if (!pairs) {
1142:         print_error ("Insufficient memory to begin list.");
1143:         return (1);
1144:         }
1145:     pairs->next = NULL;
1146:
1147:             /*** create array of images in memory ***/
1148:     temp1[0] = toupper(filename[strlen(filename)-5]);
1149:     for (ptr=(set_ptr[0])->next; ptr!=NULL; ptr=ptr->next)
1150:         for (index=0; index<num; index++)
1151:             for (last=(set_ptr[index])->next; last!=NULL; last=last->next) {
1152:                 p1 = malloc (sizeof(list));
1153:                 p2 = malloc (sizeof(list));
1154:                 if (!p1 || !p2) {
```

```
1155:                   print_error ("Insufficient memory to create image list.");
1157:                   ptr = NULL;
1158:                   index = num;
1159:                   last = NULL;
1160:                   break;
1161:                   }
1162:               p2->next = pairs->next;
1163:               p1->next = p2;
1164:               pairs->next = p1;
1165:               switch (temp1[0]) {
1166:                 case 'P':
1167:                   sprintf(p1->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d %d",
1168:                           IMAGE_DIR, ptr->filename, ptr->orient, ptr->freq, ptr->phase,
1169:                           IMAGE_DIR, last->filename, last->orient, last->freq, last->phase, LEFT);
1170:                   sprintf(p2->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d %d",
1171:                           IMAGE_DIR, last->filename, last->orient, last->freq, last->phase,
1172:                           IMAGE_DIR, ptr->filename, ptr->orient, ptr->freq, ptr->phase, RIGHT);
1173:                   break;
1174:                 case 'B':
1175:                   sprintf(p1->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d %d %d %d",
1176:                           IMAGE_DIR, ptr->filename, ptr->orient, ptr->freq, ptr->phase,
1177:                           IMAGE_DIR, last->filename, last->orient, last->freq, last->phase,
1178:                           LEFT, ptr->level, last->level),
1179:                   sprintf(p2->s, "%s%sL.IMG %d %d %d %s%s..IMG %d %d %d %d %d %d",
1180:                           IMAGE_DIR, last->filename, last->orient, last->freq, last->phase,
1181:                           IMAGE_DIR, ptr->filename, ptr->orient, ptr->freq, ptr->phase,
1182:                           RIGHT, last->level, ptr->level);
1183:                   break;
1184:                 default:
1185:                   sprintf(p1->s, "%s%s.IMG %d %d %d %s%s.IMG %d %d %d %d",
1186:                           IMAGE_DIR, ptr->filename, ptr->orient, ptr->freq, ptr->phase,
1187:                           IMAGE_DIR, last->filename, last->orient, last->freq, last->phase, LEFT);
1188:                   sprintf(p2->s, "%s%s.IMG %d %d %d %s%s.IMG %d %d %d %d",
1189:                           IMAGE_DIR, last->filename, last->orient, last->freq, last->phase,
1190:                           IMAGE_DIR, ptr->filename, ptr->orient, ptr->freq, ptr->phase, RIGHT);
1191:                 }  /* switch */
1192:               }  /* for */
1193:
1194:             /*** delete linked list ***/
1195:     for (index=0; index<num; index++)
1196:         for (last=ptr=set_ptr[index]; ptr!=NULL; ) {
1197:             ptr = last->next;
1198:             free(last);
1199:             last = ptr;
1200:             }  /* for */
1201:
1202:             /*** write array to disk and free memory ***/
1203:     sprintf(temp1, "%s%s", EXE_DIR, TEMP_FILE);
1204:     if ((file_ptr=fopen(temp1,"at"))==NULL) {
1205:         print_error("Unable to open temp file.");
1206:         for (p1=p2=pairs; p1; ) {
1207:             p1 = p1->next;
1208:             free(p2);
1209:             p2 = p1;
```

```
1210:             }
1212:         return (1);
1213:         }
1214:     for (p1=pairs->next; p1; p1=p1->next)
1215:         fprintf(file_ptr, "%s\n", p1->s);
1216:     fclose(file_ptr);
1217:     for (p1=p2=pairs->next; p1; ) {
1218:         p1 = p1->next;
1219:         free(p2);
1220:         p2 = p1;
1221:         }
1222:     pairs->next = NULL;
1223:
1224:             /*** read in complete array from disk and save in list ***/
1225:     if ((file_ptr=fopen(temp1, "rt"))==NULL) {
1226:         print_error("Unable to open temp file.");
1227:         return (1);
1228:         }
1229:     for (size=0; !feof(file_ptr); ) {
1230:         if (!fgets(pairs->s, sizeof(string), file_ptr))
1231:             break;
1232:         p1 = malloc(sizeof(list));
1233:         if (!p1) {
1234:             print_error ("Insufficient memory to load in all trials.");
1235:             fclose (file_ptr);
1236:             for (p1=pairs; p1; ) {
1237:                 p1 = p1->next;
1238:                 free(pairs);
1239:                 pairs = p1;
1240:                 }
1241:             return (1);
1242:             }
1243:         p1->next = pairs;
1244:         pairs = p1;
1245:         size++;
1246:         }
1247:     fclose(file_ptr);
1248:
1249:             /*** randomize array ***/
1250:     randomize();
1251:     for (num=0; num<NUM_PASSES; num++)
1252:         for (p1=pairs->next; p1; p1=p1->next) {
1253:             index = random(size);
1254:             for (p2=pairs->next; index-- && p2; )
1255:                 p2 = p2->next;
1256:             strcpy (temp1, p2->s);
1257:             strcpy (p2->s, p1->s);
1258:             strcpy (p1->s, temp1);
1259:             }
1260:
1261:             /*** write array to disk and free memory ***/
1262:     sprintf(temp1, "%s%s", EXE_DIR, TEMP_FILE);
1263:     if ((file_ptr=fopen(temp1, "wt"))==NULL) {
1264:         print_error("Unable to open temp file.");
```

```
1265:         for (p1=pairs; p1; ) {
1267:             p1 = p1->next;
1268:             free(pairs);
1269:             pairs = p1;
1270:             }
1271:         return (1);
1272:         }
1273:     for (p1=pairs->next; p1; p1=p1->next)
1274:         fprintf(file_ptr, "%s", p1->s);
1275:     fclose(file_ptr);
1276:     for (p1=pairs; p1; ) {
1277:         p1 = p1->next;
1278:         free(pairs);
1279:         pairs = p1;
1280:         }
1281:     return (0);
1282:
1283:     }   /* randomize_trials */
1284: /*-------------------------------------------------------------------------*/
1285:
1286:
1287:
1288: /*-------------------------------------------------------------------------*/
1289: static void set_group_sequence(void)
1290:
1291:     /*      This module creates the information used by the randomize
1292:         procedure.  Specifically, the user is asked for the group names.    */
1293:
1294:     {
1295:     FILE    *file_out;                  /* output file                  */
1296:     char    group_name[MAX_GROUP_CHAR]; /* name of group                */
1297:     int     num_groups;                 /* number of groups in file     */
1298:     int     subject_number;             /* subject's code number        */
1299:     char    string[100];                /* temporary string             */
1300:
1301:
1302:     print_title("GEXPT 2: Set Group Sequence\n\n");
1303:
1304:     get_input("Enter subject number:", "%d", &subject_number);
1305:     sprintf(string, "%ssub1%4.4d.dat", EXE_DIR, subject_number);
1306:     file_out = fopen(string, "wt");
1307:     if (file_out==NULL)
1308:         print_system_error();
1309:     else {
1310:         get_input("Enter number of groups:", "%d", &num_groups);
1311:         fprintf(file_out, "%d 1\n", num_groups);
1312:         for (; num_groups!=0; num_groups--) {
1313:             get_input("Enter group name:", "", group_name);
1314:             fprintf(file_out, "%s\n", group_name);
1315:             }
1316:         fclose(file_out);
1317:         }
1318:
1319:     }   /* set_group_sequence */
```

```
1320: /*-------------------------------------------------------------------------*/
```

```
 1: /*****************************************************************************
 2:
 3:         FILENAME:               gexpt2c.h
 4:         PROGRAMMER:             Christopher Voltz - UDRI
 5:         CREATED:                -1/8901.09
 6:         LAST MODIFIED:          -1/8901.09
 7:         INTERFACE PROTOCALL:    Turbo C 2.0
 8:         USAGE:                  #include <gexpt2c.h>
 9:
10:
11:         This file contains the header(s) for the following routine(s):
12:
13:     display_calibration_menu => This module is responsible for displaying the
14:         options for the calibration menu.  A keystroke is then read and control
15:         is transferred to the appropriate routine.
16:
17: *****************************************************************************/
18:
19:
20:
21:     /****************
22:      * .DEFINTIONS  *
25:      ****************/
24:
25: #if !defined(_BYTE)
26: #define _BYTE
27: typedef unsigned char  byte;    /* define a byte (8-bit) type */
28: #endif
29:
30:
31:
32:     /*************************
33:      *  FUNCTION PROTOTYPES  *
34:      *************************/
35:
36: byte display_calibration_menu(void);
```

```
/*****************************************************************************

        FILENAME:           gexpt2c.c
        PROGRAMMER:         Christopher Voltz - UDRI
        CREATED:            -1/8708.11
        LAST MODIFIED:      -1/8904.18
        INTERFACE PROTOCOL: Turbo C 2.0


        MODULE PURPOSES:

    calculate_lut =>    This module calculates the new LUT based upon the
        minimum and maximum ideal readings and the actual readings.  It simply
        searches through the array of actual readings for the closest value to
        each element in the array of ideal readings.  The number of the element
        in the actual array which corresponds to the element in the ideal
        array is stored in the LUT array, ie. if we are searching for the
        closest reading to element 5 in the ideal array and we find that the
        value in element 6 of the actual array is closest to the value in
        element 5 of the ideal array, then element 5 in the LUT array will be
        initialized to 6.

    display_calibration_menu => This module is responsible for displaying the
        options for this menu level.  A keystroke is then read and control is
        transferred to the appropriate routine.  It also takes care of making
        sure the LUT array is up to date relative to the ideal and actual
        arrays.  Additionally, it is responsible for ensuring that the user
        does not forget to save the calibration information if it has changed.

    get_ideal_calibration =>    This module prompts the user for the minimum
        and maximum ideal readings and creates an array whose elements are
        a linear function between those to endpoints, eg. if the OUT_LUT_SIZE is
        256, the minimum is 0, and the maximum is 255 then the ideal array
        will be initialized to 0 to 255 in steps of 1.

    load_calibration_file =>    This module prompts the user for the name of
        the file containing the calibration data.  The data is then read into
        the ideal and actual arrays.  Additionally, the minimum and maximum
        ideal readings are read.

    print_calibration_tables => This module prompts the user to see if he
        would like to print the LUT table, the ideal calibration array, or
        the actual calibration array.  This allows the user to determine
        how closely the linearization was done.

    recalibrate_monitor =>  This module prompts the user for new ideal
        minimum and maximum readings.  Then it prompts the user for the
        intensity to display.  The screen is then initialized to that value
        and the user is prompted for the spotmeter reading.  A series of
        readings taken in this fashion allows an array to be constructed which
        represents the gamma function of the monitor.  Gaps between readings
        are filled using extrapolation between the last two points taken.
        Thus, more readings will result in a more accurate gamma function.

FILE=GEXPT2C.C    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
 55:     save_calibration_file =>    This module prompts the user for the filename
 57:         to save the calibration data in.  It then writes out a line for each
 58:         element in the actual, ideal, and LUT arrays.  Finally, it writes the
 59:         minimum and maximum ideal readings.
 60:
 61:         The calibration data file consists of the calibration data used
 62:     to linearize the data.  It is stored in ASCII format with
 63:     each line containing the actual calibration reading (single
 64:     precision), the ideal calibration reading (single precision),
 65:     and the intensity value the readings correspond to (integer).
 66:     The last line contains the maximum and minimum ideal readings
 67:     (single precision).  The file is delimited by the EOF mark.
 68:
 69: ************************************************************************/
 70:
 71:
 72:
 73:
 74:     /*************
 75:      *  HEADERS  *
 76:      *************/
 77:
 78: #include <conio.h>
 79: #include <ctype.h>
 80: #include <limits.h>
 81: #include <stdio.h>
 82: #include <stdlib.h>
 83: #include <string.h>
 84:
 85:
 86:
 87:     /**************
 88:      *  INCLUDES  *
 89:      **************/
 90:
 91: #include <constant.h>
 92: #if DT2871
 93: #    include <dt2871.h>              /* routines to control DT-2871 board    */
 94: #else
 95: #    include <pcwrap.h>              /* wrapper for pcvision routines        */
 96: #endif
 97: #include <gexpt2.h>
 98: #include <gexpt2c.h>
 99: #include <graphics.h>
100: #include <toolbox.h>
101:
102:
103:
104:     /*************************
105:      *  FUNCTION PROTOTYPES  *
106:      *************************/
107:
108: void calculate_lut(float actual[], float calibrate[], int lut[]);
109: void get_ideal_calibration(float calibrate[], float *min_read, float *max_read);

FILE=GEXPT2C.C    Fri Jun 16 01:58:16 1989    PAGE=2
```

```
110: void graph_calibration(float actual[], float calibrate[], int lut[]);
112: void load_calibration_file(float actual[], float calibrate[], int lut[],
113:                            float *min_read, float *max_read);
114: void print_calibration_tables(float actual[], float calibrate[], int lut[],
115:                               float min_read, float max_read);
116: void recalibrate_monitor(float actual[], float calibrate[], float *min_read,
117:                          float *max_read);
118: void save_calibration_file(float actual[], float calibrate[], int lut[],
119:                            float min_read, float max_read);
120:
121:
122:
123:     /***************
124:      *  FUNCTIONS  *
125:      ***************/
126:
127: /*-------------------------------------------------------------------------*/
128: void calculate_lut(float actual[], float calibrate[], int lut[])
129:
130:     /*      This module attempts to find the closest actual calibration, for
131:         each LUT entry, which approximates the ideal LUT entry.             */
132:
133: {
134:     int closest;
135:     int index;
136:     int index_2;
137:
138:     for (index=0; index<OUT_LUT_SIZE; index++) {
139:         closest = 0;
140:         for (index_2=0; index_2<OUT_LUT_SIZE; index_2++)
141:             if (abs(calibrate[index]-actual[index_2]) <
142:                 abs(calibrate[index]-actual[closest]))
143:                 closest = index_2;
144:         lut[index] = closest;
145:         }
146:
147: }   /* calculate_lut */
148: /*-------------------------------------------------------------------------*/
149:
150:
151:
152: /*-------------------------------------------------------------------------*/
153: byte display_calibration_menu(void)
154:
155:     /*      This module displays the calibration menu, gets a keystroke,
156:         executes the appropriate function, and returns either a non-zero
157:         value if the menu should be displayed again, or a zero value if the
158:         user selected the exit option.  */
159:
160:     {
161:     float calibrate[OUT_LUT_SIZE];  /* ideal readings at each intensity    */
162:     float actual[OUT_LUT_SIZE];     /* actual readings at each intensity   */
163:     int   lut[OUT_LUT_SIZE];        /* conversion lookup table             */
164:
```

```
165:     float   min_read;              /* minimum ideal reading              */
167:     float   max_read;              /* maximum ideal reading              */
168:     int     option;                /* option user has chosen             */
169:
170:     enum {UNITIALIZED, SAVED, LOADED};  /* calibration info status types    */
171:     static byte status=SAVED;           /* calibration info status          */
172:
173:
174:         /*** setup the screen ***/
175:     print_title("GEXPT 2: Calibration Menu\n\n");
176:
177:         /*** print options ***/
178:     print_option("C|Change ideal readings");
179:     print_option("L|Load calibration file");
180:     print_option("P|Print calibration tables");
181:     print_option("G|Graph calibration readings");
182:     print_option("R|Recalibrate monitor");
183:     print_option("S|Save calibration file");
184:     printf("\n");
185:     print_option("x|Exit menu");
186:     print_option("<ESC>|<ESC> Exit menu");
187:     cprintf("\r\n\n%*cEnter Option: ", ENTRY_INDENT, ' ');
188:
189:         /*** get the user's option ***/
190:     textcolor(ENTRY_COLOR);
191:     switch (toupper(getche())) {
192:         case 'C': cprintf("\r\n\n");
193:                   get_ideal_calibration(calibrate, &min_read, &max_read);
194:                   cprintf("\r\n%*cCalculating...", ENTRY_INDENT, ' ');
195:                   calculate_lut(actual, calibrate, lut);
196:                   status &= ~SAVED;
197:                   break;
198:         case 'G': if (status & LOADED)
199:                     graph_calibration(actual, calibrate, lut);
200:                   else
201:                     print_error("Calibration information has not been loaded yet.");
202:                   break;
203:         case 'L': load_calibration_file(actual, calibrate, lut, &min_read, &max_read);
204:                   status |= SAVED | LOADED;
205:                   break;
206:         case 'P': print_calibration_tables(actual, calibrate, lut, min_read, max_read);
207:                   break;
208:         case 'R': recalibrate_monitor(actual, calibrate, &min_read, &max_read);
209:                   calculate_lut(actual, calibrate, lut);
210:                   status = (status & ~SAVED) | LOADED;
211:                   break;
212:         case 'S': save_calibration_file(actual, calibrate, lut, min_read, max_read);
213:                   status |= SAVED;
214:                   break;
215:         case ESC_KEY:   cprintf("\bX");
216:         case'X': if (status & SAVED)
217:                     return(EXIT_MENU);          /* exit this menu */
218:                  else {                         /* check if user forgot to save data */
219:                    cprintf("\r\n\n\n");
```

```
                    get_input("Did you wish to exit without saving the new calibration (Y/N):",
                              "%c", &option);
                    if (toupper(option)=='Y')   /* exit if he didn't */
                        return(EXIT_MENU);
                    else
                        break;                  /* otherwise, stay here */
                    }
        default: cprintf("\a");
                 break;
        }   /* switch */

        return(REDISPLAY);      /* redisplay menu */

    }   /* display_calibration_menu */
/*--------------------------------------------------------------------------*/



/*--------------------------------------------------------------------------*/
void get_ideal_calibration(float calibrate[], float *max_read, float *min_read)

{
    int index;


    cprintf("\r\n");
    get_input("Enter the minimum and maximum ideal readings:", "%f,%f",
              min_read, max_read);
    for (index=0; index<OUT_LUT_SIZE; index++)
        calibrate[index] = (*max_read - *min_read) / OUT_LUT_SIZE * index;

}   /* get_ideal_calibration */
/*--------------------------------------------------------------------------*/



/*--------------------------------------------------------------------------*/
void graph_calibration(float actual[], float calibrate[], int lut[])

    /*      This module plots the ideal gamma function and the "actual"
        gamma function as determined by the data input while taking readings.
        The curves are plotted on the screen using different lines types.
        The screen can be printed if an EGA screen dump utility has been
        activated.                                                          */

    {
    int     index;              /* general purpose index            */
    int     index_2;            /* second general purpose index     */
    char    string[50];         /* general string variable          */
    float   x_scale;            /* scaling factor for x direction   */
    float   y_max;              /* maximum value in y direction     */
    float   y_min;              /* minimum value in y direction     */
    float   y_scale;            /* scaling factor for y direction   */

FILE=GEXPT2C.C    Fri Jun 16 01:58:16 1989    PAGE=5
```

```
275:     const   X_MAX=550;
277:     const   X_MIN=100;
278:     const   Y_MAX=300;
279:     const   Y_MIN=10;
280:
281:
282:         /*** setup screen and get name of input file ***/
283:     print_title("GEXPT 2: Graph Calibration\n\n");
284:
285:         /*** initialize graphics hardware ***/
286:     if (registerfarbgidriver(EGAVGA_driver_far) < 0) {
287:         print_error("Graphics driver could not be registered.");
288:         return;
289:         }
290:     index = EGA;
291:     index_2 = EGAHI;
292:     initgraph(&index, &index_2, NULL);
293:
294:         /*** setup screen ***/
295:     cleardevice();
296:
297:         /*** draw graph axes starting in upper left going counterclockwise ***/
298:     setcolor(WHITE);
299:     moveto(X_MIN-1, Y_MIN-1);
300:     lineto(X_MIN-1, Y_MAX+1);
301:     lineto(X_MAX+1, Y_MAX+1);
302:     lineto(X_MAX+1, Y_MIN-1);
303:     lineto(X_MIN-1, Y_MIN-1);
304:
305:         /*** determine minimum and maximum of y direction ***/
306:     y_max = 0.0;
307:     y_min = 100000.0
308:     for (index=0; index<OUT_LUT_SIZE; index++) {
309:         y_max = max(y_max, actual[index]);
310:         y_max = max(y_max, calibrate[index]);
311:         y_min = min(y_min, actual[index]);
312:         y_min = min(y_min, calibrate[index]);
313:         }
314:
315:         /*** determine scaling factors ***/
316:     x_scale = (X_MAX-X_MIN)/(float)OUT_LUT_SIZE;
317:     y_scale = (Y_MAX-Y_MIN-1)/(float)(y_max-y_min);
318:
319:         /*** print graph labels ***/
320:     settextjustify(RIGHT_TEXT, CENTER_TEXT);
321:     itoa(y_min, string, 10);
322:     outtextxy(X_MIN-5, Y_MAX-5, string);
323:     itoa(y_max, string, 10);
324:     outtextxy(X_MIN-5, Y_MIN+5, string);
325:     outtextxy(X_MIN+6, Y_MAX+10, "0");
326:     itoa(OUT_LUT_SIZE-1, string, 10);
327:     outtextxy(X_MAX-5, Y_MAX+10, string);
328:     outtextxy(X_MIN-5, 150, "Reading");
329:     settextjustify(CENTER_TEXT, CENTER_TEXT);
```

```
330:     outtextxy((X_MIN+X_MAX)/2, Y_MAX+20, "LUT Index");
332:
333:         /*** plot ideal gamma function ***/
334:     setcolor(LIGHTRED);
335:     setlinestyle(DOTTED_LINE, 0, THICK_WIDTH);
336:     moveto(X_MIN, Y_MAX);
337:     for (index=0; index<OUT_LUT_SIZE; index++)
338:         lineto(X_MIN+x_scale*index, Y_MAX-y_scale*(calibrate[index]-y_min));
339:     line(X_MAX+10, Y_MIN+10, X_MAX+20, Y_MIN+10);
340:     settextjustify(LEFT_TEXT, CENTER_TEXT);
341:     outtextxy(X_MAX+25, Y_MIN+10, "Ideal");
342:
343:         /*** plot actual gamma function ***/
344:     setcolor(LIGHTBLUE);
345:     setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
346:     moveto(X_MIN, Y_MAX);
347:     for (index=0; index<OUT_LUT_SIZE; index++)
348:         lineto(X_MIN+x_scale*index, Y_MAX-y_scale*(actual[index]-y_min));
349:     line(X_MAX+10, Y_MIN+30, X_MAX+20, Y_MIN+30);
350:     outtextxy(X_MAX+25, Y_MIN+30, "Actual");
351:
352:         /*** wait for user to press key ***/
353:     getch();
354:
355:         /*** setup screen ***/
356:     cleardevice();
357:
358:         /*** draw graph axes starting in upper left going counterclockwise ***/
359:     setcolor(WHITE);
360:     setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
361:     moveto(X_MIN-1, Y_MIN-1);
362:     lineto(X_MIN-1, Y_MAX+1);
363:     lineto(X_MAX+1, Y_MAX+1);
364:     lineto(X_MAX+1, Y_MIN-1);
365:     lineto(X_MIN-1, Y_MIN-1);
366:
367:         /*** determine scaling factors ***/
368:     x_scale = (X_MAX-X_MIN)/(float)OUT_LUT_SIZE;
369:     y_scale = (Y_MAX-Y_MIN)/(float)OUT_LUT_SIZE;
370:
371:         /*** print graph labels ***/
372:     settextjustify(RIGHT_TEXT, CENTER_TEXT);
373:     itoa(0, string, 10);
374:     outtextxy(X_MIN-5, Y_MAX-5, string);
375:     itoa(255, string, 10);
376:     outtextxy(X_MIN-5, Y_MIN+5, string);
377:     outtextxy(X_MIN+6, Y_MAX+10, "0");
378:     itoa(OUT_LUT_SIZE-1, string, 10);
379:     outtextxy(X_MAX-5, Y_MAX+10, string);
380:     outtextxy(X_MIN-5, 150, "LUT value");
381:     settextjustify(CENTER_TEXT, CENTER_TEXT);
382:     outtextxy((X_MIN+X_MAX)/2, Y_MAX+20, "LUT Index");
383:
384:         /*** plot LUT ***/
```

```
385:     setfillstyle(SOLID_FILL, LIGHTBLUE);
387:     for (index=0; index<OUT_LUT_SIZE; index++)
388:         bar(X_MIN+x_scale*index, Y_MAX-y_scale*lut[index],
389:             X_MIN+x_scale*index+x_scale, Y_MAX);
390:
391:         /*** wait for user to press key ***/
392:     getch();
393:
394:         /*** shut down graphics system and restore CRT mode ***/
395:     closegraph();
396:
397:     }   /* graph_calibration */
398: /*-------------------------------------------------------------------------*/
399:
400:
401:
402: /*-------------------------------------------------------------------------*/
403: void load_calibration_file(float actual[], float calibrate[], int lut[],
404:                            float *min_read, float *max_read)
405:
406: {
407:     FILE    *file_ptr;
408:     int     index;
409:     char    string[100];
410:     char    string_2[100];
411:
412:
413:     cprintf("\r\n\n\n");
414:     string[0]=EXPAND_CHAR;
415:     string[1]=NULL;
416:     strcat(string, INIT_INFO);
417:     sprintf(string_2, "Enter calibration filename to load (%s):", string);
418:     get_input(string_2, "%s", string);
419:     if (string[0]==EXPAND_CHAR) {
420:         strcpy(string_2, EXE_DIR);
421:         strcat(string_2, string+1);
422:         strcpy(string, string_2);
423:         }
424:     strcat(string, ".CAL");
425:     if ((file_ptr=fopen(string, "rt"))==NULL) {
426:         sprintf(string_2, "Unable to open %s\n", string);
427:         print_error(string_2);
428:         return;
429:         }
430:
431:     for (index=0; index<OUT_LUT_SIZE; index++)
432:         fscanf(file_ptr, "%E,%E,%d", &actual[index], &calibrate[index], &lut[index]);
433:     fscanf(file_ptr, "%E,%E", max_read, min_read);
434:
435:     fclose(file_ptr);
436:
437: }   /* load_calibration_file */
438: /*-------------------------------------------------------------------------*/
439:
```

```
440:
442:
443: /*-------------------------------------------------------------------------*/
444: void print_calibration_tables(float actual[], float calibrate[], int lut[],
445:                               float min_read, float max_read)
446:
447: {
448:     int     index;
449:     char    response;
450:
451:
452:     print_title("GEXPT 2: Pring Calibration Tables\n\n");
453:
454:     get_input("Would you like to print the LUT (Y/N):", "%c", &response);
455:     if (toupper(response)=='Y') {
456:         fprintf(stdprn, "\n**** Intensity Remapping (LUT) Table ****\n");
457:         for (index=0; index<OUT_LUT_SIZE; index++)
458:             fprintf(stdprn, "Index %3d mapped to %3d\n", index, lut[index]);
459:         fprintf(stdprn, "\f");
460:         }
461:
462:     get_input("Would you like to print the Ideal Calibration Table (Y/N):",
463:               "%c", &response);
464:     if (toupper(response)=='Y') {
465:         fprintf(stdprn, "\n**** Ideal Calibration Table ****\n");
466:         for (index=0; index<OUT_LUT_SIZE; index++)
467:             fprintf(stdprn, "Intensity %3d => Reading = %6.2f\n", index, calibrate[index]);
468:         fprintf(stdprn, "\n\nMin intensity=%f, Max intensity=%f\f", min_read, max_read);
469:         }
470:
471:     get_input("Would you like to print the Actual Calibration Table (Y/N):",
472:               "%c", &response);
473:     if (toupper(response)=='Y') {
474:         fprintf(stdprn, "\n**** Actual Calibration Table ****\n");
475:         for (index=0; index<OUT_LUT_SIZE; index++)
476:             fprintf(stdprn, "Intensity %3d => Reading = %6.2f\n', index, actual[index]);
477:         fprintf(stdprn, "\f");
478:         }
479:
480: }   /* print_calibration_tables */
481: /*-------------------------------------------------------------------------*/
482:
483:
484:
485: /*-------------------------------------------------------------------------*/
486: void recalibrate_monitor(float actual[], float calibrate[], float *min_read,
487:                          float *max_read)
488:
489: {
490:     int     index;
491:     int     intensity;
492:     int     old_intensity;
493:     float   reading;
494:

FILE=GEXPT2C.C    Fri Jun 16 01:58:16 1989    PAGE=9
```

```
495:
497:        /*** setup screen ***/
498:    print_title("GEXPT 2: Recalibrate Monitor");
499:
500:        /*** initialize the graphics hardware ***/
501:    cprintf("\r\n\n%*cInitializing the board prior to calibration.\r\n",
502:            ENTRY_INDENT, ' ');
503:    initialize_hardware();
504:
505:        /*** get the ideal calibration readings ***/
506:    get_ideal_calibration(calibrate, min_read, max_read);
507:
508:        /*** refresh the screen ***/
509:    print_title("GEXPT 2: Recalibrate Monitor\r\n");
510:
511:        /*** print the data entry instructions ***/
512:    cprintf("\r\n     Enter the intensity number you wish to display on the\r\n"
513:            "monitor and press enter.  Then enter the intensity reading from\r\n"
514:            "the spotmeter and press return.  Take readings starting from 0\r\n"
515:            "and increase in whatever steps you like.\r\n"
516:            "     When all the readings have been entered enter -1 as the intensity\n\r"
517:            "and press enter.\r\n"
518:            "     To simply abort the calibration, enter -2 as the intensity and\r\n"
519:            "press enter.\r\n\n"
520:            "NOTE: the intensity numbers must be entered in increasing\n\r"
521:            "      numerical order.  This allows the program to fill in the\n\r"
522:            "      gaps between readings by interpolating.  Also the remaining\n\r"
523:            "      numbers in the LUT are filled with extrapolated values so\n\r"
524:            "      readings close to the maximum intensity will result in a\n\r"
525:            "      better correction.\r\n\n"
526:            "NOTE: the new calibration will not be used until the board is\n\r"
527:            "      initialized again.\r\n\n");
528:
529:        /*** get the actual calibration readings and extrapolate between ***
530:         *** endpoints                                                   ***/
531:    window(1, wherey(), 80, wherey()+2);
532:    old_intensity = intensity = index = 0;
533:    while (intensity!=-1 && intensity!=-2) {
534:        clrscr();
535:        get_input("Enter intensity:", "%d", &intensity);
536:        if (intensity<0)
537:            break;
538:        display_buffer(0);
539:        clear_screen(intensity, 0);
540:        get_input("Enter reading:", "%f", &reading);
541:        if (intensity>=0) {
542:            actual[intensity] = reading;
543:            for (index=1; index<(intensity-old_intensity); index++)
544:                actual[index+old_intensity] = (reading-actual[old_intensity]) /
545:                                              (intensity-old_intensity)*index +
546:                                              actual[old_intensity];
547:            old_intensity = intensity;
548:            }   /* else if */
549:        }   /* while */

FILE=GEXPT2C.C    Fri Jun 16 01:58:16 1989    PAGE=10
```

```
550:
552:        /*** linearly extend the last point to fill the rest of the array ***/
553:    window(1, 1, 80, 25);
554:    gotoxy(ENTRY_INDENT, 23);
555:    textcolor(MENU_COLOR);
556:    if (intensity==-2)  /* if just used for testing, exit */
557:        return;
558:    cprintf(" Calculating...");
559:    reading = actual[old_intensity] - actual[old_intensity-1];
560:    for (index=1; index<(OUT_LUT_SIZE-old_intensity); index++)
561:        actual[index+old_intensity] = reading*index + actual[old_intensity];
562:
563: }   /*  recalibrate_monitor */
564: /*-------------------------------------------------------------------------*/
565:
566:
567:
568: /*-------------------------------------------------------------------------*/
569: void save_calibration_file(float actual[], float calibrate[], int lut[],
570:                            float min_read, float max_read)
571:
572: {
573:    FILE    *file_ptr;
574:    int     index;
575:    char    string[100];
576:    char    string_2[100];
577:
578:
579:    cprintf("\r\n\n\n");
580:    string[0]=EXPAND_CHAR;
581:    string[1]=NULL;
582:    strcat(string, INIT_INFO);
583:    sprintf(string_2, "Enter calibration filename to save (%s):", string);
584:    get_input(string_2, "%s", string);
585:    if (string[0]==EXPAND_CHAR) {
586:        strcpy(string_2, EXE_DIR);
587:        strcat(string_2, string+1);
588:        strcpy(string, string_2);
589:        }
590:    strcat(string, ".CAL");
591:    if ((file_ptr=fopen(string, "wt"))==NULL) {
592:        sprintf(string_2, "Unable to open %s\n", string);
593:        print_error(string_2);
594:        return;
595:        }
596:
597:    for (index=0; index<OUT_LUT_SIZE; index++)
598:        fprintf(file_ptr, "%E, %E, %d\n", actual[index], calibrate[index], lut[index]);
599:    fprintf(file_ptr, "%E, %E\n", max_read, min_read);
600:
601:    fclose(file_ptr);
602:
603: }   /* save_calibration_file */
604: /*-------------------------------------------------------------------------*/
```

```
/*****************************************************************************

            FILENAME:           gexpt2d.h
            PROGRAMMER:         Christopher Voltz - UDRI
            CREATED:            -1/8901.17
            LAST MODIFIED:      -1/8903.15
            INTERFACE PROTOCOL: TURBO C 2.0
            USAGE:              #include <gexpt2d.h>


        This module contains the headers for the following routine(s):

    display_response_data_collection_menu   ->  This routine is responsible
        for displaying the constant stimuli menu.  It displays the options
        available, records a keystroke, and executes the appropriate
        function.  It will return a value indicating whether the menu
        should redisplayed (called again) or not.

*****************************************************************************/



/**********************
 * FUNCTION PROTOTYPES *
 **********************/

byte display_response_data_collection_menu (void);
```

```
/*****************************************************************************

            FILENAME:               gexpt2d.c
            PROGRAMMER:             Christopher Voltz - UDRI
            CREATED:                -1/8901.17
            LAST MODIFIED:          -1/8904.20
            INTERFACE PROTOCOL:     TURBO C 2.0
            USAGE:                  #include <gexpt2d.h>


        This module contains code for the following routines:

    analyze_data    -> This routine analyzes the constant stimuli data and
        writes a summary to the stream passed to it.  If the stream is a NULL
        pointer, then the user is prompted for a stream to write the summary
        data to.

    collect_data    -> This routine presents stimuli, collects the data, and
        produces .RAW and .SUM files.  The stimuli are presented using the
        constant stimuli method.  The response box is used for data input.

    display_response_data_collection_menu  -> This routine allows the user
        to choose, from a menu, whether he would like to: set the group
        presentation sequence, or test the response box, or collect data, or
        analyze data, or graph a summary file.

    flash_screens   -> This routine displays each screen for a specified
        duration and then proceeds to the next screen in the sequence.  A
        sample presentation might be: noise, adapting, noise, stimulus, mask,
        and noise.

    graph_summary_file  -> This routine graphs the summary file created by
        the analyze_data routine.

    randomize_trials    -> This routine creates the file which the
        collect_data routine reads.  The file consists of the filenames of
        images to be displayed and their relevant data, ie. phase,
        orientation, frequency, etc.

    set_group_sequence ->  This routine determines the selection and order
        the groups are presented in.

*****************************************************************************/



    /****************
     * HEADER FILES *
     ****************/

    /* TURBO C header files */
#include <alloc.h>
#include <conio.h>
#include <ctype.h>
```

```
 55: #include <dos.h>
 57: #include <graphics.h>
 58: #include <math.h>
 59: #include <stdio.h>
 60: #include <stdlib.h>
 61: #include <string.h>
 62: #include <time.h>
 63:
 64:     /* program specific header files */
 65: #include <constant.h>           /* program constants to define system   */
 66:                                 /* parameters for included files        */
 67: #if DT2871
 68: #    include <dt2871.h>        /* routines to control DT-2871 board    */
 69: #else
 70: #    include <pcwrap.h>        /* wrapper for pcvision routines        */
 71: #endif
 72: #include <response.h>          /* routines to control response box     */
 73: #include <toolbox.h>           /* general utility routines             */
 74: #include <gexpt2.h>            /* header file of main program          */
 75: #include <gexpt2d.h>           /* this module's header file            */
 76:
 77:
 78:
 79:     /*************************
 80:      *  FUNCTION PROTOTYPES  *
 81:      *************************/
 82:
 83: static void analyze_data(FILE *raw_file, FILE *sum_file);
 84: static void collect_data(void);
 85: static void display_group_sequence();
 86: static void flash_screens(int adapt_duration, int duration);
 87: static void graph_summary_file(void);
 88: static byte randomize_trials(char *filename);
 89: static void set_group_sequence(void);
 90:
 91:
 92:
 93:     /*************************
 94:      *  FUNCTION DEFINITIONS  *
 95:      *************************/
 96:
 97:
 98:
 99: /*-------------------------------------------------------------------------*/
100: static void analyze_data(FILE *raw_file, FILE *sum_file)
101:
102: {
103:     struct node_
104:     {
105:         int             freq;       /* number of frequencies in image       */
106:         int             orient;     /* number of orientations in image      */
107:         int             level;      /* bandwith level, if used              */
108:         int             count;      /* num of x => n                        */
109:         int             num_same;   /* number of same responses             */

FILE=GEXPT2D.C    Fri Jun 16 01:58:16 1989    PAGE=2
```

```
110:        struct node_    *next;      /* link to next node in list           */
112:    };
113:    typedef struct node_ node;
114:
115:    boolean     automatic;          /* analysis following data collection? */
116:    boolean     bandwith=' ';       /* indicates if bandwith analysis      */
117:    int         band_level_c=0;     /* bandwith level of comparison        */
118:    int         band_level_s=0;     /* bandwith level of standard          */
119:    struct date date_now;           /* current date                        */
120:    char        filename[60];       /* filename of file to open            */
121:    boolean     found;              /* indicates if node currently exists  */
122:    int         num_freq_c;         /* number of frequencies in comparison */
123:    int         num_freq_s;         /* number of frequencies in standard   */
124:    int         num_orient_c;       /* number of orientations in comparison */
125:    int         num_orient_s;       /* number of orientations in standard  */
126:    node        *occur;             /* pointer to lists of nodes           */
127:    node        *ptr;               /* pointer in lists of nodes           */
128:    int         response;           /* response by subject                 */
129:    int         same;               /* if left and right images are same   */
130:    char        string[100];        /* temporary string                    */
131:    node        *temp_ptr;          /* temporary pointer                   */
132:    struct time time_now;           /* current time                        */
133:
134:
135:        /* print title screen */
136:    print_title("GEXPT 2: Analyze Data\n\n");
137:
138:        /* determine if analysis is part of data collection or not */
139:    automatic = (sum_file!=NULL);
140:
141:        /* determine filename of output file if necessary */
142:    if (!automatic)
143:    {
144:        get_input("Enter filename of output file (.SUM):", "%s", string);
145:        if (string[0]==EXPAND_CHAR)
146:            sprintf(filename, "%s%s.SUM", DATA_DIR, string+1);
147:        else
148:            sprintf(filename, "%s.SUM", string);
149:        if ((sum_file=fopen(filename, "wt"))==NULL)
150:        {
151:            print_error("Unable to open output file");
152:            return;
153:        }
154:        fprintf(sum_file, "GEXPT 2 Same/Different Stimuli Summary File: \t\s\n",
155:                filename);
156:        getdate(&date_now);
157:        gettime(&time_now);
158:        fprintf(sum_file, "CREATED: %d/%d/%d at %02d:%02d:%02d\n",
159:                date_now.da_mon, date_now.da_day, date_now.da_year,
160:                time_now.ti_hour, time_now.ti_min, time_now.ti_sec);
161:        fprintf(sum_file, "%s consists of:");
162:    }
163:
164:        /* prepare to prompt user if not in automatic mode */
```

```
    if (automatic)
        textcolor(BLACK);
    else
        textcolor(MENU_COLOR);
    cprintf("%*cHit CR to end filename entry.\n\r\n", ENTRY_INDENT, ' ');
    occur = NULL;

            /* for each group, analyze file */
    for (strcpy(filename, "CV"); strlen(filename)!=0; )
    {

            /* determine filename and open file if necessary */
        if (!automatic)
        {
            get_input("Enter name of raw data file to analyze (.RAW):", "",
                    filename);
            if (strlen(filename)==0)
                continue;
            if (filename[0]==EXPAND_CHAR)
                sprintf(string, "%s%s.RAW", DATA_DIR, filename+1);
            else
                sprintf(string, "%s.RAW", filename);
            if ((raw_file=fopen(string, "rt"))==NULL)
            {
                print_error("Unable to find specified file.");
                continue;
            }
            fprintf(sum_file, "%*c%s\n", ENTRY_INDENT, ' ', string);
        }
        else
            filename[0] = 0;

            /* check to see if file is correct format */
        fgets(string, 100, raw_file);
        if (!strcmp(string, "4\n"))
            if (bandwith==' ')
                bandwith = TRUE;
            else if (bandwith==FALSE)
            {
                print_error("All files to be analyzed must be of the same type.");
                fclose(raw_file);
                fclose(sum_file);
                return;
            }
        else if (!strcmp(string, "3\n"))
            if (bandwith==' ')
                bandwith = FALSE;
            else if (bandwith==TRUE)
            {
                print_error("All files to be analyzed must be of the same type.");
                fclose(raw_file);
                fclose(sum_file);
                return;
            }
```

```
220:        else
222:        {
223:            print_error("File is not a same/different raw data file.");
224:            fclose(raw_file);
225:            fclose(sum_file);
226:        }
227:
228:            /* read in data and append to current list */
229:        while (!feof(raw_file))
230:        {
231:            fgets(string, 100, raw_file);
232:            if (bandwith)
233:                sscanf(string, "%*d %*d %d %d %d %d %*d %*d %d %d %d %d %*d %*d "
234:                       "%*d %d %d", &num_orient_s, &num_freq_s, &num_orient_c,
235:                       &num_freq_c, &response, &same, &band_level_s,
236:                       &band_level_c);
237:            else
238:                sscanf(string, "%*d %*d %d %d %d %d %*d %*d %d %d %d %d %*d %*d "
239:                       "%*d", &num_orient_s, &num_freq_s, &num_orient_c,
240:                       &num_freq_c, &response, &same);
241:            for (ptr=occur, found=FALSE; ptr!=NULL && !found;)
242:            {
243:                found = (ptr->freq==num_freq_c && ptr->orient==num_orient_c &&
244:                         ptr->level==band_level_c);
245:                if (!found)
246:                    ptr = ptr->next;
247:            }
248:            if (!found)
249:            {
250:                if ((ptr=(node *)malloc(sizeof(node)))==NULL)
251:                {
252:                    print_error("Insufficient memory.");
253:                    for (ptr=occur; ptr!=NULL; ptr=occur)
254:                    {
255:                        occur = occur->next;
256:                        free(ptr);
257:                    }
258:                    fclose(raw_file);
259:                    fclose(sum_file);
260:                    return;
261:                }
262:                ptr->next = occur;
263:                ptr->level = band_level_c;
264:                ptr->freq = num_freq_c;
265:                ptr->orient = num_orient_c;
266:                ptr->num_same = ptr->count = 0;
267:                occur = ptr;
268:            }
269:            if (response)   /*  responded same?  */
270:                ptr->num_same ++;
271:            ptr->count ++;
272:        }   /* while */
273:        fclose(raw_file);
274:

FILE=GEXPT2D.C    Fri Jun 16 01:58:16 1989    PAGE=5
```

```
275:     }   /* for */
277:
278:         /* sort nodes by orientation, and frequency */
279:     for (ptr=occur; ptr!=NULL; ptr=ptr->next)
280:         for (temp_ptr=ptr; temp_ptr!=NULL; temp_ptr=temp_ptr->next)
281:             if ( (ptr->freq<temp_ptr->freq) ||
282:                  ((ptr->freq==temp_ptr->freq) && (ptr->orient<temp_ptr->orient)) ||
283:                  ((ptr->freq==temp_ptr->freq) && (ptr->orient==temp_ptr->orient) &&
284:                   (ptr->level>temp_ptr->level)) )
285:             {
286:                 swap_int(&(ptr->level),     &(temp_ptr->level));
287:                 swap_int(&(ptr->orient),    &(temp_ptr->orient));
288:                 swap_int(&(ptr->freq),      &(temp_ptr->freq));
289:                 swap_int(&(ptr->count),     &(temp_ptr->count));
290:                 swap_int(&(ptr->num_same),  &(temp_ptr->num_same));
291:             }
292:
293:         /* write # correct and # trials and close file */
294:     if (bandwith)
295:         fprintf(sum_file, "\n%.2s  %.2s  %.2s  %.3s  %.8s\n",
296:                 "SF", "OR", "BW", " # Same ", "# Trials");
297:     else
298:         fprintf(sum_file, "\n%.2s  %.2s  %.8s  %.8s\n",
299:                 "SF", "OR", " # Same ", "# Trials");
300:     for (ptr=occur; ptr!=NULL; ptr=ptr->next)
301:         if (bandwith)
302:             fprintf (sum_file, "%2d  %2d  %2d  %8d  %8d\n",  ptr->freq,
303:                     ptr->orient, ptr->level, ptr->num_same, ptr->count);
304:         else
305:             fprintf (sum_file, "%2d  %2d  %8d  %8d\n",  ptr->freq, ptr->orient,
306:                     ptr->num_same, ptr->count);
307:
308:     fclose(sum_file);
309:
310:         /* free nodes and return */
311:     for (ptr=occur; ptr!=NULL; )
312:         {
313:         temp_ptr = ptr->next;
314:         free (ptr);
315:         ptr = temp_ptr;
316:         }
317:
318: }   /* analyze_data */
319: /*-------------------------------------------------------------------------*/
320:
321:
322:
323: /*-------------------------------------------------------------------------*/
324: static void collect_data(void)
325:
326:     /*
327:             This module is responsible for presenting the stimulus pairs,
328:         reading the subject's response, recording all data in the output
329:         file, and writing the summary of the data.
```

```
330:     */
331:
332:
333: {
334: #if ADAPT
335:     int         adapt_duration; /* number of fields to display adapting */
336: #endif
337:     boolean     bandwith=' ';   /* indicates if bandwith stimuli in use */
338:     struct date date_now;       /* current date                         */
339:     int         duration;       /* number of fields to display images   */
340:     int         eccentricity;   /* how far out center of image is       */
341:     char        filename[60];   /* filename of output file              */
342:     FILE        *file_in;       /* input file pointer                   */
343:     FILE        *file_ptr;      /* general file pointer                 */
344:     header_type header;         /* header for image files               */
345:     int         index;          /* general loop control variable        */
346:     int         index_2;        /* second general loop control variable */
347:     int         level_1=0;      /* bandwidth level in image 1           */
348:     int         level_2=0;      /* bandwidth level in image 2           */
349:     int         num_freq_1;     /* number of frequencies in image 1     */
350:     int         num_freq_2;     /* number of frequencies in image 2     */
351:     int         num_groups;     /* number of groups to run in session   */
352:     int         num_orient_1;   /* number of orientations in image 1    */
353:     int         num_orient_2;   /* number of orientations in image 2    */
354:     int         phase_1;        /* phase number of image 1              */
355:     int         phase_2;        /* phase number of image 2              */
356:     FILE        *raw_file;      /* raw data file pointer                */
357:     int         response;       /* subject's response                   */
358:     int         session;        /* session number                       */
359:     int         side;           /* side that signal is on               */
360:     char        string[100];    /* temp variable                        */
361:     char        string_2[100];  /* temp variable                        */
362:     int         subject_num;    /* subject number                       */
363:     FILE        *sum_file;      /* summary file pointer                 */
364:     struct time time_now;       /* current time                         */
365:     int         val;            /* temp variable                        */
366:
367:
368:             /* setup screen */
369:     print_title("GEXPT 2: Subject Data Entry\n\n");
370:
371:             /* initialize graphics hardware */
372:     initialize_hardware();
373:
374:             /* get data to create filename */
375:     subject_num = session = eccentricity = duration = num_groups = 0;
376:     get_input("Subject number:", "%d", &subject_num);
377:     get_input("Session number:", "%d", &session);
378:     get_input("Eccentricity (1=0.75 deg, 2=20 deg):", "%d", &eccentricity);
379:     get_input("Stimulus duration (0=167 ms, 1=334 ms):", "%d", &duration);
380:     if (duration)
381:         duration = 20;  /* 20 fields => 10 frames => 334 ms */
382:     else
383:         duration = 10;  /* 10 fields => 5 frames => 167 ms  */
384:     get_input("Number of groups to run (0 to abort):", "%d", &num_groups);
```

```
385:    if (!num_groups)
387:        return;
388:
389:            /* open data files */
390:    sprintf(filename, "%s%04.4d%04.4d.SUM", DATA_DIR, subject_num, session);
391:    if ((sum_file=fopen(filename, "wt"))==NULL)
392:    {
393:        print_system_error();
394:        return;
395:    }
396:    fprintf(sum_file, "GEXPT 2 Same/Different Stimuli Summary File: %s\t\n",
397:            filename);
398:
399:    sprintf(filename, "%s%04.4d%04.4d.RAW", DATA_DIR, subject_num, session);
400:    if ((raw_file=fopen(filename, "w+t"))==NULL)
401:    {
402:        print_system_error();
403:        goto error_exit_1;
404:    }
405:
406:            /* save header in data file */
407:    getdate(&date_now);
408:    gettime(&time_now);
409:    fprintf(sum_file, "CREATED: %d/%d/%d at %02d:%02d:%02d\n",
410:            date_now.da_mon, date_now.da_day, date_now.da_year,
411:            time_now.ti_hour, time_now.ti_min, time_now.ti_sec);
412:
413:            /* get noise screen filename */
414:    filename[0] = 0;
415:    get_input("Enter noise screen filename (``###.IMG  CR for none):", "%s",
416:              filename);
417:    if (filename[0]!=0)
418:    {
419:        strcpy(string, IMAGE_DIR);
420:        strcat(string, "G");
421:        strcat(string, filename);
422:        strcpy(filename, string);
423:        fprintf(sum_file, "Noise screen: %s\n", filename);
424:        strcat(filename, "GL.IMG");
425:        read_image(NOISE_BUFFER, LEFT, filename, header);
426:        strcat(string, "GR.IMG");
427:        read_image(NOISE_BUFFER, RIGHT, string, header);
428:    }
429:
430:            /* get adapting screen filename */
431:    filename[0] = 0;
432:    get_input("Enter adapting screen filename (``###.IMG  CR for none):", "%s",
433:              filename);
434:    if (filename[0]==0)
435:        adapt_flag = FALSE;
436:    else
437:    {
438:        adapt_flag = TRUE;
439:        strcpy(string, IMAGE_DIR);
```

```
440:         strcat(string, "G");
442:         strcat(string, filename);
443:         strcpy(filename, string);
444:         fprintf(sum_file, "Adapting screen filename: %s\n", filename);
445:         strcat(filename, "GL.IMG");
446:         read_image(ADAPT_BUFFER, LEFT, filename, header);
447:         strcat(string, "GR.IMG");
448:         read_image(ADAPT_BUFFER, RIGHT, string, header);
449: #if ADAPT
450:         get_input("Adapting duration (0=167 ms, 1=334 ms):", "%d",
451:                 &adapt_duration);
452:         if (adapt_duration)
453:             adapt_duration = 20;    /* 20 fields => 10 frames => 334 ms */
454:         else
455:             adapt_duration = 10;    /* 10 fields => 5 frames => 167 ms  */
456: #endif
457:     }
458:
459:             /* remove old randomized trial file */
460:     sprintf(string, "%s%s", EXE_DIR, TEMP_FILE);
461:     unlink(string);
462:
463:             /* remind user to turn on response box */
464:     cprintf("\r\n\n%*cRemember to turn on response box......now randomizing.\n",
465:             ENTRY_INDENT, ' ');
466:
467:             /* for each group repeat the following */
468:     while (num_groups--)
469:     {
470:
471:             /*
472:                 open group sequence file and read current group;
473:                 also update current group pointer
474:             */
475:         sprintf(string, "%ssub2%4.4d.DAT", EXE_DIR, subject_num);
476:         if ((file_in=fopen(string, "rt"))==NULL)
477:         {
478:             print_system_error();
479:             goto error_exit_2;
480:         }
481:         sprintf(string, "%s%s", EXE_DIR, TEMP_FILE_2);
482:         if ((file_ptr=fopen(string, "wt"))==NULL)
483:         {
484:             print_system_error();
485:             goto error_exit_3;
486:         }
487:         fscanf(file_in, "%d %d\n", &val, &index_2);
488:         fprintf(file_ptr, "%d %d\n", val, (++index_2 > val ? 1 : index_2));
489:         for (index=1; index<index_2 && !feof(file_in); index++)
490:         {
491:             fgets(filename, 60, file_in);
492:             fprintf(file_ptr, "%s", filename);
493:         }
494:         for (; index<=val && !feof(file_in); index++)
```

```
495:        {
497:            fgets(string, 100, file_in);
498:            fprintf(file_ptr, "%s", string);
499:        }
500:        fclose(file_ptr);
501:        fclose(file_in);
502:        sprintf(string, "%ssub2%4.4d.DAT", EXE_DIR, subject_num);
503:        sprintf(string_2, "%s%s", EXE_DIR, TEMP_FILE_2);
504:        unlink(string);
505:        rename(string_2, string);
506:
507:            /* open group directory and determine set filename */
508:        sprintf(string, "%s%s", EXE_DIR, GROUP_DIR);
509:        if ((file_in=fopen(string, "rt"))==NULL)
510:        {
511:            print_system_error();
512:            goto error_exit_2;
513:        }
514:        fscanf(file_in, "%*d\n");   /* skip highest group number */
515:        for (strcpy(string, ""); strcmp(string, filename) && !feof(file_in); )
516:        {
517:            fgets(string, 100, file_in);    /* read group name  */
518:            fgets(string_2, 100, file_in);  /* read filename    */
519:        }
520:        if (strcmp(string, filename))
521:        {
522:            print_error("Could not find specified group.");
523:            goto error_exit_3;
524:        }
525:        fclose(file_in);
526:
527:            /* create randomized list of trial info */
528:        string_2[strlen(string_2)-1] = '\0';        /* kill CR */
529:        sprintf(string, "%s%s", EXE_DIR, string_2);
530:        if (bandwith==' ')
531:            bandwith = toupper(string_2[strlen(string_2)-5]) == 'B';
532:        else
533:            if ((toupper(string_2[strlen(string_2)-4])=='B') && !bandwith)
534:            {
535:                print_error ("All groups must be either bandwith or not bandwith.");
536:                goto error_exit_2;
537:            }
538:        if (randomize_trials(string))
539:            goto error_exit_2;
540:    }   /*  while  */
541:
542:        /* wait for subject to signal ready */
543:    textcolor(MENU_COLOR);
544:    cprintf("\n\r\n%*cPress any response key to begin.\n\r\n",
545:            ENTRY_INDENT, ' ', ENTRY_INDENT, ' ');
546:    if (!get_response())
547:        goto error_exit_2;
548:
549:        /* display sequence of trials; get and record responses */
```

```
550:     sprintf(string, "%s%s", EXE_DIR, TEMP_FILE);
552:     if ((file_in=fopen(string, "rt"))==NULL)
553:     {
554:         print_error("Unable to open temp file.");
555:         goto error_exit_2;
556:     }
557:     if (!bandwith)
558:         fprintf (raw_file, "3\n");
559:     else
560:         fprintf (raw_file, "4\n");
561:     while (!feof(file_in))
562:     {
563:         fscanf(file_in, "%s %d %d %d %s %d %d %d %d", string,
564:                &num_orient_1, &num_freq_1, &phase_1, string_2,
565:                &num_orient_2, &num_freq_2, &phase_2, &side);
566:         if (bandwith)
567:             fscanf (file_in, "%d %d\n", &level_1, &level_2);
568:         else
569:             fscanf (file_in, "\n");
570:         if (bandwith)
571:             cprintf("\n\r%*c'%s' %2d %2d %2d %1d  '%s' %2d %2d %2d %1d  %5s\n\r",
572:                     ENTRY_INDENT, ' ', string, num_freq_1, num_orient_1,
573:                     phase_1, level_1, string_2, num_freq_2, num_orient_2,
574:                     phase_2, level_2, (side?"Left ":"Right"));
575:         else
576:             cprintf("\n\r%*c'%s' %2d %2d %2d  '%s' %2d %2d %2d   %5s\n\r",
577:                     ENTRY_INDENT, ' ', string, num_freq_1, num_orient_1,
578:                     phase_1, string_2, num_freq_2, num_orient_2, phase_2,
579:                     (side?"Left ":"Right"));
580:         if (side==RIGHT)
581:         {
582:             swap_int(&level_1, &level_2);
583:             swap_int(&num_orient_1, &num_orient_2);
584:             swap_int(&num_freq_1, &num_freq_2);
585:             swap_int(&phase_1, &phase_2);
586:         }
587:         display_buffer(ADAPT_BUFFER);
588:         read_2_images(SIGNAL_BUFFER, string, header, LEFT, string_2, header);
589:         flash_screens(0 /* adapt duration */, duration);
590:         cprintf("%*cEnter response: ", ENTRY_INDENT, ' ');
591:         textcolor(ENTRY_COLOR);
592:         switch (response=get_response())
593:         {
594:             case R_BUTTON_3:
595:             case R_BUTTON_4:
596:                 cprintf("Different");
597:                 response = 0;
598:                 break;
599:             case R_BUTTON_1:
600:             case R_BUTTON_2:
601:                 cprintf("Same");
602:                 response = 1;
603:                 break;
604:         }  /* switch */
```

```
605:         if (kbhit())
607:         {
608:             getch();
609:             break;
610:         }
611:         textcolor(MENU_COLOR);
612:         cprintf("\n\r");
613:         fprintf(raw_file, "%1d %1d  %2d %2d  %2d %2d  %1d %1d %1d %2d %1d "
614:                 "%1d %1d %1d ", subject_num, session, num_orient_1,
615:                 num_freq_1, num_orient_2, num_freq_2, side, eccentricity,
616:                 response, num_freq_2*num_orient_2,
617:                 (int)(strnicmp(string, string_2, strlen(string)-5)==0 ? 1 : 0),
618:                 (int)(duration==10 ? 0 : 1), phase_1, phase_2);
619:         if (bandwith)
620:             fprintf (raw_file, " %1d %1d\n", level_1, level_2);
621:         else
622:             fputc ('\n', raw_file);
623:     }   /* while */
624:
625:         /* inform user it is end of group */
626:     sound(BEEP_FREQUENCY);
627:     delay(BEEP_DELAY/2);
628:     nosound();
629:     delay(BEEP_DELAY/2);
630:     sound(BEEP_FREQUENCY);
631:     delay(BEEP_DELAY/2);
632:     nosound();
633:
634:         /* analyze the data and return */
635:     rewind(raw_file);
636:     analyze_data(raw_file, sum_file);
637:     return;
638:
639:         /*
640:              control is transferred here only if an error occurs; multiple
641:              entry points allow the cleanup code to eliminate duplicate
642:              code
643:          */
644: error_exit_4:
645:             fclose(file_ptr);
646: error_exit_3:
647:             fclose(file_in);
648: error_exit_2:
649:             fclose(raw_file);
650: error_exit_1:
651:             fclose(sum_file);
652: }   /* collect_data */
653: /*--------------------------------------------------------------------------*/
654:
655:
656:
657: /*--------------------------------------------------------------------------*/
658: static void display_group_sequence(void)
659:
```

```
660:     /*      This module displays the information used by the randomize
661:         procedure.  Specifically, the group names are displayed.      */
662:
663:     {
664:     FILE    *file_in;                   /* input file                  */
665:     int     current_group;              /* current group number       */
666:     char    group_name[MAX_GROUP_CHAR]; /* name of group              */
667:     int     num_groups;                 /* number of groups in file   */
668:     int     subject_number;             /* subject's code number      */
669:     char    string[100];                /* temporary string           */
670:
671:
672:     print_title("GEXPT 2: Display Group Sequence\n\n");
673:
674:     get_input("Enter subject number:", "%d", &subject_number);
675:     sprintf(string, "%ssub2%4.4d.dat", EXE_DIR, subject_number);
676:     file_in = fopen(string, "rt");
677:     if (file_in==NULL)
678:         print_system_error();
679:     else {
680:         print_title("GEXPT 2: Display Group Sequence\n\n");
681:         fscanf(file_in, "%d %d\n", &num_groups, &current_group);
682:         cprintf("Subject %-d:\r\n", subject_number);
683:         cprintf("    has %-d groups.\r\n", num_groups);
684:         cprintf("    will use group number %-d next.\r\n", current_group);
685:         cprintf("\r\nThe groups are:\r\n");
686:         for (current_group=1; num_groups!=0; num_groups--, current_group++) {
687:             fgets(group_name, MAX_GROUP_CHAR, file_in);
688:             cprintf("%*c(%-d)  %s\r", ENTRY_INDENT, ' ', current_group,
689:                     group_name);
690:             }
691:         fclose(file_in);
692:         cprintf("\r\n\nPress any key to continue.");
693:         getch();
694:         }
695:
696:     }   /* display_group_sequence */
697: /*-------------------------------------------------------------------------*/
698:
699:
700: /*-------------------------------------------------------------------------*/
701: byte display_response_data_collection_menu(void)
702:
703:     /*
704:         This module displays the constant stimuli menu, gets a keystroke,
705:         executes the appropriate function, and returns either a non-zero
706:         value if the menu should be displayed again, or a zero value if the
707:         user selected the exit option.
708:     */
709:
710: {
711:
712:         /* setup the screen */
713:
```

```
715:     print_title("GEXPT 2: Same/Different Data Collect/Analyze Menu\n\n");
716:
717:         /* print options */
718:     print_option("A|Analyze data file");
719:     print_option("C|Collect data");
720:     print_option("D|Display group sequence");
721:     print_option("G|Graph summary file");
722:     print_option("S|Set group presentation sequence");
723:     print_option("T|Test response box");
724:     printf("\n");
725:     print_option("x|Exit menu");
726:     print_option("<ESC>|<ESC> Exit menu");
727:     cprintf("\r\n\n%*cEnter Option: ", ENTRY_INDENT, ' ');
728:
729:         /* get the user's option */
730:     textcolor(ENTRY_COLOR);
731:     switch (toupper(getche()))
732:     {
733:         case 'A':
734:             analyze_data(NULL, NULL);
735:             break;
736:         case 'C':
737:             collect_data();
738:             break;
739:         case 'D':
740:             display_group_sequence();
741:             break;
742:         case 'G':
743:             graph_summary_file();
744:             break;
745:         case 'S':
746:             set_group_sequence();
747:             break;
748:         case 'T':
749:             test_response_box();
750:             break;
751:         case ESC_KEY:
752:             cprintf("\bX");
753:         case'X':
754:             return(EXIT_MENU);  /* exit this menu */
755:     }
756:
757:     return(REDISPLAY);  /* redisplay menu */
758:
759: }   /* display_menu */
760: /*---------------------------------------------------------------------------*/
761:
762:
763:
764: /*---------------------------------------------------------------------------*/
765: static void flash_screens(int adapt_duration, int duration)
766:
767:     /*
768:             This module flashes the stimulus screen on for a period of
769:
```

```
770:         time, preceeded by either a noise screen, or an adapting screen
772:         followed the noise screen; and followed by the noise screen.
773:     */
774:
775: {
776:
777:     screen_hold(1);                          /* sync screen changes to   */
778:                                              /* vertical interrupt       */
779: #if ADAPT
780:     if (adapt_flag)
781:     {
782:         display_buffer(ADAPT_BUFFER);        /* show adapting harmonic   */
783:         screen_hold(adapt_duration);         /*   for # fields           */
784:     }
785:     else
786:         display_buffer(NOISE_BUFFER);
787: #endif
788:     display_buffer(NOISE_BUFFER);
789:     screen_hold(0.25 *2*1000/16.7);
790:
791:     display_buffer(SIGNAL_BUFFER);           /* show stimulus        */
792:     screen_hold(duration);                   /*   for # fields       */
793:
794:     display_buffer(NOISE_BUFFER);
795:
796: #pragma warn -par
797: }   /* flash_screens */
798: #pragma warn .par
799: /*-------------------------------------------------------------------------*/
800:
801:
802:
803: /*-------------------------------------------------------------------------*/
804: static void graph_summary_file(void)
805:
806:     /*
807:           This module reads in the summary data from a specified file and
808:         plots it on the screen using different line types.  The screen can
809:         be printed if an EGA screen dump utility has been activated.
810:     */
811:
812: {
813:     boolean     bandwith;            /* indicates if bandwith levels in use  */
814:     FILE        *file_in;            /* summary file pointer                 */
815:     char        filename[1[illegible];/* filename of summary file            */
816:     int         frequency;           /* frequency of current line            */
817:     int         last_frequency;      /* last frequency used                  */
818:     int         last_orientation;    /* last orientation used                */
819:     int         level;               /* bandwith level of current line       */
820:     int         line_type;           /* indicates current line type          */
821:     int         num_same;            /* number of same responses             */
822:     int         num_lines;           /* number of sets displayed             */
823:     int         num_trials;          /* number of trials in set              */
824:     int         orientation;         /* orientation of current line          */
```

```
825:    char        temp[100];          /* used to expand filename          */
827:    int         x, y;               /* current graphics position        */
828:
829:
830:        /* setup screen and get name of input file */
831:    print_title("GEXPT 2: Graph Summary Files\n\n");
832:    get_input("Enter name of file to graph (.SUM):", "%s", filename);
833:    if (filename[0]==EXPAND_CHAR)
834:    {
835:        strcpy(temp, DATA_DIR);
836:        strcat(temp, filename+1);
837:        strcpy(filename, temp);
838:    }
839:    strcat(filename, ".SUM");
840:
841:        /* open file */
842:    if ((file_in=fopen(filename, "rt"))==NULL)
843:    {
844:        print_error("Could not open graph file.");
845:        return;
846:    }
847:
848:        /* see if file is of correct type */
849:    fgets(temp, 100, file_in);
850:    if (strncmp(temp, "GEXPT 2 Same/Different Stimuli Summary File:", 44))
851:    {
852:        print_error("Not a same/different summary file.");
853:        fclose (file_in);
854:        return;
855:    }
856:    for (; temp[0]!='\n'; fgets(temp, 100, file_in))
857:        ;
858:    fgets(temp, 100, file_in);
859:    bandwith = !strncmp("SF  OR  BW", temp, 10);
860:
861:        /* initialize graphics hardware */
862:    if (registerfarbgidriver(EGAVGA_driver_far) < 0) {
863:        print_error("Graphics driver could not be registered.");
864:        fclose(file_in);
865:        return;
866:        }
867:    orientation = EGA;
868:    frequency = EGAHI;
869:    initgraph(&orientation, &frequency, NULL);
870:
871:        /* setup screen */
872:    cleardevice();
873:
874:        /* draw graph axes */
875:    setcolor(WHITE);
876:    moveto(100, 15);
877:    lineto(100, 305);
878:    lineto(550, 305);
879:    lineto(550, 15);
```

```
880:     lineto(100, 15);
882:
883:         /* print graph labels */
884:     for (frequency=100; frequency>-1; frequency-=10)
885:     {
886:         itoa(frequency, temp, 10);                          /*  label Y axis  */
887:         outtextxy(70, 12+(100-frequency)*2.9, temp);
888:     }
889:     settextjustify(BOTTOM_TEXT, CENTER_TEXT);
890:     settextstyle(DEFAULT_FONT, VERT_DIR, 1);
891:     outtextxy(40, 145, "Percent Same");
892:
893:     settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
894:     settextjustify(CENTER_TEXT, TOP_TEXT);
895:     if (bandwith)                                           /*  label X axis  */
896:         for (frequency=1; frequency<6; frequency++)
897:         {
898:             itoa(frequency, temp, 10);
899:             outtextxy(101+frequency*74.667, 315, temp);
900:         }
901:     else
902:         for (frequency=1; frequency<9; frequency++)
903:         {
904:             itoa(frequency*8, temp, 10);
905:             outtextxy(94+frequency*50, 315, temp);
906:         }
907:     if (bandwith)
908:         outtextxy(320, 335, "Bandwith Level");
909:     else
910:         outtextxy(320, 335, "Number of Components");
911:     outtextxy(320, 0, filename);
912:     settextjustify(RIGHT_TEXT, TOP_TEXT);
913:
914:     frequency = orientation = last_frequency = last_orientation =
915:                 level = num_lines = 1;
916:     line_type = -1;
917:     while (!feof(file_in))
918:     {
919:         fgets(temp, 100, file_in);
920:         if (bandwith)
921:             sscanf(temp, "%d %d %d %d %d", &frequency, &orientation, &level,
922:                     &num_same, &num_trials);
923:         else
924:             sscanf(temp, "%d %d %d %d", &frequency, &orientation, &num_same,
925:                     &num_trials);
926:         if (frequency!=last_frequency || (bandwith ? orientation!=last_orientation : 0))
927:         {
928:             switch (line_type)
929:             {
930:                 case  0: line_type = 1;
931:                          setcolor(LIGHTBLUE);
932:                          setfillstyle(SOLID_FILL, LIGHTBLUE);
933:                          break;
934:                 case  1: line_type = 3;
```

```
                        setcolor(LIGHTCYAN);
                        setfillstyle(SOLID_FILL, LIGHTCYAN);
                        break;
                case -1:
                case  3: line_type = 0;
                        setcolor(LIGHTRED);
                        setfillstyle(SOLID_FILL, LIGHTRED);
                        break;
            }   /* switch */
            itoa(frequency, temp, 10);
            outtextxy(585, 25+num_lines*10, temp);
            if (bandwith)
            {
                itoa(orientation, temp, 10);
                outtextxy(610, 25+num_lines*10, temp);
            }
            setlinestyle(line_type, 0, THICK_WIDTH);
            line(625, 28+num_lines*10, 639, 28+num_lines*10);
            moveto(bandwith ? (101+level*74.667) : (101+frequency*orientation/8.0*49.778),
                    16+(1.0-num_same/(float)num_trials)*288);
            num_lines++;
        }
        else
            lineto(bandwith ? (101+level*74.667) : (101+frequency*orientation/8.0*49.778),
                    16+(1.0-num_same/(float)num_trials)*288);
        x = getx();
        y = gety();
        fillellipse(bandwith ? (101+level*74.667) : (101+frequency*orientation/8.0*49.778),
                    16+(1.0-num_same/(float)num_trials)*288,
                    2, 2);
        moveto(x, y);
        last_frequency = frequency;
        last_orientation = orientation;
    }
    fclose(file_in);

        /* wait for user to press key */
    getch();

        /* shut down graphics system and restore CRT mode */
    closegraph();

}   /* graph_summary_file */
/*-------------------------------------------------------------------------*/



/*-------------------------------------------------------------------------*/
static byte randomize_trials(char *filename)

    /*
            This module reads in the the set info in the file given by
        filename; creates a file containing all the valid possible
        combinations of set A with each of the other sets (on an image by
```

```
 990:         image basis); randomizes the order of the entries in the file; and
 992:         returns to the caller.
 993:     */
 994:
 995: {
 996:     struct node_struct
 997:     {
 998:         char                filename[60];
 999:         int                 freq;
1000:         int                 orient;
1001:         int                 phase;
1002:         int                 level;
1003:         struct node_struct  *next;
1004:     };
1005:     typedef struct node_struct node;
1006:     typedef char string[70];
1007:     struct list_struct
1008:     {
1009:         string              s;
1010:         struct list_struct  *next;
1011:     };
1012:     typedef struct list_struct list;
1013:
1014:     boolean     bandwith;               /* indicates if bandwith stimuli in use     */
1015:     int         bandwith_level;         /* level of bandwith                        */
1016:     FILE        *file_ptr;              /* file pointer                             */
1017:     int         index;                  /* general index variable                   */
1018:     node        *last;                  /* general pointer                          */
1019:     int         num;                    /* number of sets in use                    */
1020:     int         num_freq;               /* number of frequencies for a given set    */
1021:     int         num_orient;             /* number of orientations for a given set   */
1022:     list        *pairs;                 /* first entry in list of image pairs       */
1023:     list        *p1;                    /* general pointer                          */
1024:     list        *p2;                    /* general pointer                          */
1025:     node        *ptr;                   /* general pointer                          */
1026:     node        *set_ptr[MAX_SETS];     /* array of pointers to each set            */
1027:     int         size;                   /* size of array to be randomized           */
1028:     char        temp1[100];             /* temporary string variable                */
1029:
1030:
1031:         /* determine if bandwith stimuli in use */
1032:     bandwith = (toupper(filename[strlen(filename)-5])=='B');
1033:
1034:         /* open set file and read in set info */
1035:     if ((file_ptr=fopen(filename, "rt"))==NULL)
1036:     {
1037:         print_system_error();
1038:         return (1);
1039:     }
1040:     fscanf(file_ptr, "%d ", &num);
1041:
1042:         /*
1043:             allocate space for each head pointer and initialize each
1044:             next pointer

FILE=GEXPT2D.C    Fri Jun 16 01:58:16 1989    PAGE=19
```

```
1045:         */
1047:     for (index=0; index<MAX_SETS; index++)
1048:     {
1049:         if ((set_ptr[index]=(node *)malloc(sizeof(node)))==NULL)
1050:         {
1051:             print_error("Insufficient memory.");
1052:             goto error_exit_1;
1053:         } /* if */
1054:         (set_ptr[index])->next = NULL;
1055:     }
1056:
1057:         /*
1058:             read in the sets and add the set entry to the linked list of
1059:             entries
1060:         */
1061:     for (index=size=0; index<num && !feof(file_ptr); index++)
1062:     {
1063:         fscanf(file_ptr, "%d %d ", &num_orient, &num_freq);
1064:         if (bandwith)
1065:             fscanf(file_ptr, "%d", &bandwith_level);
1066:         fscanf(file_ptr, "\n");
1067:         for (last=set_ptr[index], strcpy(temp1, ""); !feof(file_ptr)
1068:              && strcmp(temp1, "."); last=ptr)
1069:         {
1070:             if ((ptr=(node *)malloc(sizeof(node)))==NULL)
1071:             {
1072:                 print_error("Insufficient memory to read in set info.");
1073:                 goto error_exit_1;
1074:             }
1075:             fgets(temp1, 100, file_ptr);
1076:             temp1[strlen(temp1)-1] = '\0';  /* kill CR */
1077:             if (strcmp(temp1, "."))
1078:             {
1079:                 sscanf(temp1, "%s %d", ptr->filename, &(ptr->phase));
1080:                 ptr->freq = num_freq;
1081:                 ptr->orient = num_orient;
1082:                 ptr->level = (bandwith ? bandwith_level : 0);
1083:                 ptr->next = last->next;
1084:                 last->next = ptr;
1085:                 size++;
1086:             }
1087:         }
1088:     }
1089:     fclose(file_ptr);
1090:
1091:             /* allocate space for array of image pair info */
1092:     pairs = malloc (sizeof(list));
1093:     if (!pairs)
1094:     {
1095:         print_error ("Insufficient memory to begin list.");
1096:         return (1);
1097:     }
1098:     pairs->next = NULL;
1099:
```

```
1100:           /* create array of images in memory */
1102:     temp1[0] = toupper(filename[strlen(filename)-5]);
1103:     for (ptr=(set_ptr[0])->next; ptr!=NULL; ptr=ptr->next)
1104:         for (index=0; index<num; index++)
1105:             for (last=(set_ptr[index])->next; last!=NULL; last=last->next)
1106:             {
1107:                 p1 = malloc (sizeof(list));
1108:                 p2 = malloc (sizeof(list));
1109:                 if (!p1 || !p2) {
1110:                     print_error ("Insufficient memory to create image list.");
1111:                     ptr = NULL;
1112:                     index = num;
1113:                     last = NULL;
1114:                     break;
1115:                     }
1116:                 p2->next = pairs->next;
1117:                 p1->next = p2;
1118:                 pairs->next = p1;
1119:                 switch (temp1[0])
1120:                 {
1121:                     case 'B':
1122:                         sprintf(p1->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d "
1123:                                 "%d %d %d", IMAGE_DIR, ptr->filename,
1124:                                 ptr->orient, ptr->freq, ptr->phase, IMAGE_DIR,
1125:                                 last->filename, last->orient, last->freq,
1126:                                 last->phase, LEFT, ptr->level, last->level);
1127:                         sprintf(p2->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d "
1128:                                 "%d %d %d", IMAGE_DIR, last->filename,
1129:                                 last->orient, last->freq, last->phase,
1130:                                 IMAGE_DIR, ptr->filename, ptr->orient,
1131:                                 ptr->freq, ptr->phase, RIGHT, last->level,
1132:                                 ptr->level);
1133:                         break;
1134:                     case 'P':
1135:                         sprintf(p1->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d "
1136:                                 "%d", IMAGE_DIR, ptr->filename, ptr->orient,
1137:                                 ptr->freq, ptr->phase, IMAGE_DIR,
1138:                                 last->filename, last->orient, last->freq,
1139:                                 last->phase, LEFT);
1140:                         sprintf(p2->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d "
1141:                                 "%d", IMAGE_DIR, last->filename, last->orient,
1142:                                 last->freq, last->phase, IMAGE_DIR,
1143:                                 ptr->filename, ptr->orient, ptr->freq,
1144:                                 ptr->phase, RIGHT);
1145:                         break;
1146:                     default:
1147:                         sprintf(p1->s, "%s%s.IMG %d %d %d %s%s.IMG %d %d %d "
1148:                                 "%d", IMAGE_DIR, ptr->filename, ptr->orient,
1149:                                 ptr->freq, ptr->phase, IMAGE_DIR,
1150:                                 last->filename, last->orient, last->freq,
1151:                                 last->phase, LEFT);
1152:                         sprintf(p2->s, "%s%sL.IMG %d %d %d %s%sR.IMG %d %d %d "
1153:                                 "%d", IMAGE_DIR, last->filename, last->orient,
1154:                                 last->freq, last->phase, IMAGE_DIR,
```

```
1155:                                   ptr->filename, ptr->orient, ptr->freq,
1157:                                   ptr->phase, RIGHT);
1158:                 }  /* switch */
1159:             }   /* for */
1160:
1161:             /* delete linked list */
1162:         for (index=0; index<num; index++)
1163:             for (last=ptr=set_ptr[index]; ptr!=NULL; )
1164:             {
1165:                 ptr = last->next;
1166:                 free(last);
1167:                 last = ptr;
1168:             }  /* for */
1169:
1170:             /* write array to disk and free memory */
1171:         sprintf(temp1, "%s%s", EXE_DIR, TEMP_FILE);
1172:         if ((file_ptr=fopen(temp1, "at"))==NULL)
1173:         {
1174:             print_error("Unable to open temp file.");
1175:             goto error_exit_2;
1176:         }
1177:         for (p1=pairs->next; p1; p1=p1->next)
1178:             fprintf(file_ptr, "%s\n", p1->s);
1179:         fclose(file_ptr);
1180:         for (p1=p2=pairs->next; p1; ) {
1181:             p1 = p1->next;
1182:             free(p2);
1183:             p2 = p1;
1184:             }
1185:         pairs->next = NULL;
1186:
1187:             /*** read in complete array from disk and save in list ***/
1188:         if ((file_ptr=fopen(temp1, "rt"))==NULL)
1189:         {
1190:             print_error("Unable to open temp file.");
1191:             return (1);
1192:         }
1193:         for (size=0; !feof(file_ptr); )
1194:         {
1195:             if (!fgets(pairs->s, sizeof(string), file_ptr))
1196:                 break;
1197:             p1 = malloc(sizeof(list));
1198:             if (!p1)
1199:             {
1200:                 print_error ("Insufficient memory to load in all trials.");
1201:                 fclose (file_ptr);
1202:                 goto error_exit_2;
1203:             }
1204:             p1->next = pairs;
1205:             pairs = p1;
1206:             size++;
1207:         }
1208:         fclose(file_ptr);
1209:
```

```
1210:             /* randomize array */
1212:     randomize();
1213:     for (num=0; num<NUM_PASSES; num++)
1214:         for (p1=pairs->next; p1; p1=p1->next)
1215:         {
1216:             index = random(size);
1217:             for (p2=pairs->next; index-- && p2; p2=p2->next)
1218:                 ;
1219:             strcpy (temp1, p2->s);
1220:             strcpy (p2->s, p1->s);
1221:             strcpy (p1->s, temp1);
1222:         }
1223:
1224:             /* write array to disk and free memory */
1225:     sprintf(temp1, "%s%s", EXE_DIR, TEMP_FILE);
1226:     if ((file_ptr=fopen(temp1,"wt"))==NULL)
1227:     {
1228:         print_error("Unable to open temp file.");
1229:         goto error_exit_2;
1230:     }
1231:     for (p1=pairs->next; p1; p1=p1->next)
1232:         fprintf(file_ptr, "%s", p1->s);
1233:     fclose(file_ptr);
1234:     for (p1=pairs; p1; )
1235:     {
1236:         p1 = p1->next;
1237:         free(pairs);
1238:         pairs = p1;
1239:     }
1240:     return (0);
1241:
1242:         /* if an error occurs then this cleanup code is executed */
1243: error_exit_1:
1244:     for (index=0; index<num; index++)
1245:         for (last=ptr=set_ptr[index]; ptr!=NULL; )
1246:         {
1247:             ptr = last->next;
1248:             free(last);
1249:             last = ptr;
1250:         }  /* for */
1251:     fclose(file_ptr);
1252:     return (1);
1253:
1254: error_exit_2:
1255:     for (p1=p2=pairs; p1; )
1256:     {
1257:         p1 = p1->next;
1258:         free(p2);
1259:         p2 = p1;
1260:     }
1261:     return (1);
1262:
1263: }   /* randomize_trials */
1264: /*---------------------------------------------------------------------------*/
```

```
1265:
1267:
1268:
1269: /*-------------------------------------------------------------------------*/
1270: static void set_group_sequence(void)
1271:
1272:     /*      This module creates the information used by the randomize
1273:         procedure.  Specifically, the user is asked for the group names.    */
1274:
1275:     {
1276:     FILE    *file_out;                  /* output file                      */
1277:     int     num_groups;                 /* number of groups in file         */
1278:     char    string[80];                 /* temporary string                 */
1279:     int     subject_number;             /* subject number                   */
1280:
1281:
1282:     print_title("GEXPT 2: Set Group Sequence\n\n");
1283:
1284:     get_input("Enter subject number:", "%d", &subject_number);
1285:     sprintf(string, "%ssub2%4.4d.DAT", EXE_DIR, subject_number);
1286:     file_out = fopen(string, "wt");
1287:     if (file_out==NULL)
1288:         print_system_error();
1289:     else {
1290:         get_input("Enter number of groups:", "%d", &num_groups);
1291:         fprintf(file_out, "%d 1\n", num_groups);
1292:         for (; num_groups!=0; num_groups--) {
1293:             get_input("Enter group name:", "", string);
1294:             fprintf(file_out, "%s\n", string);
1295:             }
1296:         fclose(file_out);
1297:         }
1298:
1299:     }   /* set_group_sequence */
1300: /*-------------------------------------------------------------------------*/
```

IV. LISTING OF AUXILIARY PROGRAMS

gexpt2.prj
makefile.
builtins.mak
dt2871.h
dt2871.c
386move.h
386move.asm
pcvision.h
pcvision.c
pcwrap.h
pcwrap.c
response.h
response.c
toolbox.h
toolbox.c

```
\tc\lib\response (constant.h, \tc\include\response.h)
\tc\lib\toolbox (constant.h, \tc\include\toolbox.h)
\tc\lib\pcvision (constant.h, \tc\include\toolbox.h, \tc\include\pcvision.h)
\tc\lib\pcwrap (constant.h, \tc\include\pcwrap.h)

gexpt2a (gexpt2a.h,
                gexpt2b.h,
                \tc\include\pcwrap.h,
                \tc\include\toolbox.h,
                constant.h)

gexpt2b (gexpt2b.h,
                gexpt2.h,
                \tc\include\pcwrap.h
                \tc\include\response.h,
                \tc\include\toolbox.h,
                constant.h)

gexpt2c (gexpt2c.h,
                gexpt2a.h,
                \tc\include\pcwrap.h,
                \tc\include\toolbox.h,
                constant.h)

gexpt2d (gexpt2d.h,
                gexpt2.h,
                \tc\include\pcwrap.h,
                \tc\include\response.h,
                \tc\include\toolbox.h,
                constant.h)

gexpt2 (gexpt2a.h,
                gexpt2b.h,
                gexpt2c.h,
                \tc\include\pcwrap.h,
                \tc\include\response.h,
                \tc\include\toolbox.h,
                constant.h)
```

```
  1: #
  2: #                                FILENAME:                       makefile.
  3: #                                PROGRAMMER:                     Christopher Voltz - UDRI
  4: #                                CREATED:                        -1/8901.11
  5: #                                LAST MODIFIED:          -1/8901.19
  6: #                                INTERFACE PROTOCOL:     Turbo Make 1.0
  7: #
  8: #
  9: #               This makefile requires that the appropriate TURBOC.CFG be accessable
 10: #       It is presumed that the contents of TURBOC.CFG match the environment
 11: #       settings; although, the sync command will ensure this is the case.
 12: #               Additionally, the file gexpt2.lnk must be updated to include the
 13: #       filenames of the files which need to be linked together.
 14:
 15: #
 16: #       define default directories
 17: #
 18: INC=\tc\include
 19: LIB=\tc\lib
 20:
 21: #
 22: #       define a default memory model
 23: #
 24: !if !$d(MM)
 25: MM=l
 26: !endif
 27:
 28: #
 29: #       define default library and object files for compiled C code
 30: #
 31: COBJ=$(LIB)\c0$(MM)
 32: CLIB=$(LIB)\c$(MM) $(LIB)\fp87 $(LIB)\math$(MM) $(LIB)\graphics
 33:
 34: #
 35: #       define normal options for compiler, linker, assembler, and make
 36: #
 37: OPTC=-I$(INC);. -L$(LIB);. -m$(MM) -v -c
 38: OPTL=/3/d
 39: OPTA=/ml/zi/t
 40: OPTM=-s -a
 41:
 42: #
 43: #       define options for compiler, linker, assembler, and make which make listings
 44: #
 45: !if $d(LST)
 46: OPTC=-m$(MM) -I$(INC);. -L$(LIB);. -m$(MM) -v  -S
 47: OPTL=/3/m/l/s/d
 48: OPTA=/ml/zi/la/t
 49: OPTM=-s -a -DLST
 50: !endif
 51:
 52: #
 53: #       define object and source files
 54: #
```

```
OBJ=gexpt2.obj gexpt2a.obj gexpt2b.obj gexpt2c.obj gexpt2d.obj dt2871.obj \
        toolbox.obj response.obj 386move.obj
SRC=gexpt2 gexpt2a gexpt2b gexpt2c gexpt2d dt2871 toolbox response 386move
AUX=$(LIB)\dt2871.c $(LIB)\toolbox.c $(LIB)\response.c $(LIB)\386move.asm

#
#       define dependencies
#
gexpt2.exe: $(OBJ)
        tlink $(OPTL) $(CCOBJ) @gexpt2.lnk, gexpt2, , $(CLIB)

gexpt2.obj: gexpt2.c constant.h gexpt2.h gexpt2a.h gexpt2b.h gexpt2c.h \
                        gexpt2d.h $(INC)\toolbox.h $(INC)\dt2871.h $(INC)\response.h
        tcc $(OPTC) gexpt2.c

gexpt2a.obj: gexpt2a.c constant.h gexpt2a.h gexpt2b.h $(INC)\toolbox.h \
                        $(INC)\dt2871.h
        tcc $(OPTC) gexpt2a.c

gexpt2b.obj: gexpt2b.c constant.h gexpt2b.h gexpt2.h $(INC)\toolbox.h \
                        $(INC)\dt2871.h $(INC)\response.h
        tcc $(OPTC) gexpt2b.c

gexpt2c.obj: gexpt2c.c constant.h gexpt2c.h gexpt2a.h $(INC)\toolbox.h \
                        $(INC)\dt2871.h
        tcc $(OPTC) gexpt2c.c

gexpt2d.obj: gexpt2d.c constant.h gexpt2d.h gexpt2.h $(INC)\toolbox.h \
                        $(INC)\dt2871.h $(INC)\response.h
        tcc $(OPTC) gexpt2d.c

toolbox.obj: $(LIB)\toolbox.c constant.h $(INC)\toolbox.h
        tcc $(OPTC) $(LIB)\toolbox.c

dt2871.obj: $(LIB)\dt2871.c constant.h $(INC)\dt2871.h $(INC)\toolbox.h \
                        $(INC)\386move.h
        tcc $(OPTC) $(LIB)\dt2871.c

response.obj: $(LIB)\response.c constant.h $(INC)\response.h
        tcc $(OPTC) $(LIB)\response.c

386move.obj: $(LIB)\386move.asm constant.h $(INC)\386move.h
        tasm $(OPTA) $(LIB)\386move.asm


#
#       ensure unspecified parameters are as in integrated environment
#
sync:
        tcconfig tcconfig.tc turboc.cfg

#
#       define a clean directive to clean out old files
#

FILE=MAKEFILE    Fri Jun 16 01:58:16 1989    PAGE=2
```

```
 10: clean:
112: #       clean current directory
113:         erase *.obj
114:         erase *.bak
115:         erase tmp.*
116:         erase *.tmp
117:         erase *.map
118:         erase *.lst
119: #       clean INC directory
120:         erase $(INC)\*.obj
121:         erase $(INC)\*.bak
122:         erase $(INC)\tmp.*
123:         erase $(INC)\*.tmp
124:         erase $(INC)\*.map
125:         erase $(INC)\*.lst
126:         erase $(INC)\*.exe
127: #       clean LIB directory
128:         erase $(LIB)\*.bak
129:         erase $(LIB)\tmp.*
130:         erase $(LIB)\*.tmp
131:         erase $(LIB)\*.map
132:         erase $(LIB)\*.lst
133:         erase $(LIB)\*.exe
134:         cls
135:
136: #
137: #       define a save directive to save files to disk B:
138: #
139: save:
140:         copy gexpt2?.c b:
141:         copy gexpt2?.h b:
142:         copy gexpt2.prj b:
143:         copy gexpt2.in b:
144:         copy tcconfig.tc b:
145:         copy gexpt2.bat b:
146:         copy makefile. b:
147:         copy constant.h b:
148:         copy $(INC)\response.h b:
149:         copy $(INC)\toolbox.h b:
150:         copy $(INC)\dt2871.h b:
151:         copy $(INC)\386move.h b:
152:         copy $(LIB)\response.c b:
153:         copy $(LIB)\toolbox.c b:
154:         copy $(LIB)\dt2871.c b:
155:         copy $(LIB)\386move.asm b:
156:         copy $(LIB)\graphics.lib b:
157:         cls
158:
159: #
160: # define an update directive to copy files from B: to correct directories
161: #
162: update:
163:         copy b:response.h $(INC)
164:         copy b:toolbox.h $(INC)

FILE=MAKEFILE    Fri Jun 16 01:58:16 1989    PAGE=3
```

```
165:          copy b:dt2871.h $(INC)
167:          copy b:386move.h $(INC)
168:          copy b:response.c $(LIB)
169:          copy b:toolbox.c $(LIB)
170:          copy b:dt2871.c $(LIB)
171:          copy b:386move.asm $(LIB)
172:          copy b:graphics.lib $(LIB)
173:          copy b:gexpt2?.h
174:          copy b:gexpt2?.c
175:          copy b:gexpt2.prj
176:          copy b:gexpt2.in
177:          copy b:tcpick.tcp
178:          copy b:tcconfig.tc
179:          copy b:gexpt2.bat
180:          copy b:constant.h
181:          copy b:makefile.
182:          cls
183:
184: #
185: #       define a directive to touch all files so as to cause a build
186: #
187: build:
188: #       touch the files
189:          cls
190:          touch gexpt2?.c
191:          touch $(AUX)
192:          make $(OPTM) gexpt2.exe
```

```
 1: #implicit rule to compile .c files to .obj using tcc
 2: !if $d(LST)
 3: .c.obj:
 4:         tcc $(OPTC) $<
 5:         tasm $(OPTA) $*
 6: !else
 7: .c.obj:
 8:         tcc $(OPTC) $<
 9: !endif
10:
11: #implicit rule to assemble .asm files to .obj using tasm
12: #.asm.obj:
13: #       tasm $(OPTA) $<
14:
15: #implicit rule to compile/assemble .cas files to .obj using tcc/tasm
16: #!if $d(LST)
17: #.cas.obj:
18: #       tcc $(OPTC) $<
19: #       tasm $(OPTA) $*
20: #!else
21: #.cas.obj:
22: #       tcc $(OPTC) $<
23: #!endif
```

```
/*************************************************************************

            FILENAME:               dt2871.h
            PROGRAMMER:             Christopher Voltz
            CREATED:                -1/8809.13
            LAST MODIFIED:          -1/8811.06
            INTERFACE PROTOCOL:     TURBO C 2.0
            USAGE:              program.c:
                                    #include <dt2871.h>

                                constant.h:
                                    #define _BYTE 1
                                    typedef unsigned char byte
                                    #define _WORD 1
                                    typedef unsigned short int word
                                    #define _DWORD 1
                                    typedef unsigned long int doubleword
                                    #define ENTRY_INDENT 5
                                    #define ERROR_COLOR LIGHTRED+BLINK
                                    #define MENU_COLOR  WHITE


    This module provides the routines necessary to use the Data Translation
2871 board.  Specifically, it provides: a clear routine to clear the screen
to a given intensity; an initialization routine to initialize the registers
and LUTs on the board; a LUT selection routine which sets the active LUT from
a given LUT set; a LUT write routine which allows the values of the active LUT
to be set; a routine to read in an AOI (or the entire screen) to be read from
the disk; a routine to read in two consecutive quandrants from the disk; a
routine to hold a given screen for a specified number of vertical blanking
intervals; and a routine to display the contents of the image header.
    This module is simply the header information for these routines, the
actual code is contained in DT2871.C

*************************************************************************/



    /****************************
     * PREPROCESSOR DEFINITIONS *
     ****************************/

    /*** define input, output, and result lut sizes ***/
#define IN_LUT_SIZE     512     /* input LUT has 512 entries                */
#define OUT_LUT_SIZE    256     /* output LUT has only 256 entries          */


    /*** define the number of input, output, and result LUTs ***/
#define NUM_IN_LUTS     8       /* number of input LUTs                     */
#define NUM_OUT_LUTS    8       /* number of output LUTs                    */
#define NUM_RESULT_LUTS 4       /* number of result LUTs                    */


    /*** define values which indicate the type of LUT ***/

FILE=DT2871.H    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
#define INPUT_LUT       1       /* indicates the input LUT is to be used    */
#define OUTPUT_LUT      2       /* indicates the output LUT is to be used   */
#define RESULT_LUT      3       /* indicates the result LUT is to be used   */


    /*** define size of image header ***/
#define IMAGE_HEADER_SIZE   64  /* size of header-comment                   */
#define SIZE_OF_HEADER      320 /* size of header=64 (data) + 255 (comment) */


    /*** make sure required types and constants are defined ***/
#if !defined(_BYTE)
#define _BYTE
typedef unsigned char byte
#endif

#if !defined(_WORD)
#define _WORD
typedef unsigned short int word
#endif

#if !defined(_DWORD)
#define _DWORD
typedef unsigned long int doubleword
#endif

#if !defined(ENTRY_INDENT)
#define ENTRY_INDENT 5
#endif

#if !defined(ERROR_COLOR)
#define ERROR_COLOR LIGHTRED+BLINK
#endif

#if !defined(MENU_COLOR)
#define MENU_COLOR  WHITE
#endif



    /*********************
     * TYPE DEFINITIONS *
     *********************/

    /*** define types of display states ***/
enum display_states (OFF, ON);


    /*** define the input and output LUT types  ***/
typedef byte in_lut_type[IN_LUT_SIZE];  /* make an input LUT type   */

struct out_lut_entry {                  /* make an output LUT type  */
    word    red_green;                  /* with a red/green field   */
    word    blue;                       /* and a blue field         */
```

FILE=DT2871.H Fri Jun 16 01:58:16 1989 PAGE=2

```
110:     };
112:
113: typedef struct out_lut_entry out_lut_type[OUT_LUT_SIZE];
114:
115:
116:     /*** define a general LUT type ***/
117: union general_lut_type {
118:     out_lut_type    out_lut;        /* output LUT   */
119:     in_lut_type     in_lut;         /* input LUT    */
120:     in_lut_type     result_lut;     /* result LUT   */
121:     };
122:
123: typedef union general_lut_type  lut_type;
124:
125:
126:     /*** define a type to be used for image file headers ***/
127: typedef byte header_type[SIZE_OF_HEADER];
128:
129:
130:
131:     /************************
132:      * FUNCTION PROTOTYPES *
133:      ************************/
134:
135: byte clear_screen(byte intensity, byte screen);
136: void display(byte state);
137: void display_buffer(byte buffer);
138: byte init_board(void);
139: void print_header_info(header_type header);
140: byte read_image(byte buffer, byte quad, char *pathname, header_type header);
141: byte read_2_images(byte buffer, char *pathname_1, header_type header_1,
142:                    byte quad,   char *pathname_2, header_type header_2);
143: void screen_hold(word num_fields);
144: void set_write_protect(byte value);
145: void write_lut(byte type_of_lut, byte lut_number, word start, word stop,
146:                lut_type *lut);
```

```
/*************************************************************************

            FILENAME:               dt2871.c
            PROGRAMMER:             Christopher Voltz
            CREATED:                -1/8809.13
            LAST MODIFIED:          -1/8902.21
            INTERFACE PROTOCOL:     TURBO C 2.0
            USAGE:              program.c:
                                    #include <dt2871.h>

                                constant.h:
                                    #define _BYTE 1
                                    typedef unsigned char byte
                                    #define _WORD 1
                                    typedef unsigned short int word
                                    #define _DWORD 1
                                    typedef unsigned long int doubleword
                                    #define ENTRY_INDENT 5
                                    #define ERROR_COLOR LIGHTRED+BLINK
                                    #define MENU_COLOR  WHITE


    This module provides the routines necessary to use the Data Translation
2871 board.  Specifically, it provides: a clear routine to clear the screen
to a given intensity; an initialization routine to initialize the registers
and LUTs on the board; a LUT selection routine which sets the active LUT from
a given LUT set; a LUT write routine which allows the values of the active LUT
to be set; a routine to read in an AOI (or the entire screen) to be read from
the disk; a routine to read in two consecutive quandrants from the disk; a
routine to hold a given screen for a specified number of vertical blanking
intervals; and a routine to display the contents of the image header.
    This module is the actual code for these routines, the header information
is contained in DT2871.H

*************************************************************************/



    /****************
     * HEADER FILES *
     ****************/

#include <alloc.h>
#include <conio.h>
#include <dos.h>
#include <float.h>
#include <mem.h>
#include <stdio.h>

#include <constant.h>
#include <dt2871.h>
#include <386move.h>
#include <toolbox.h>
```

```
55:
57:
58:
59:
60:     /****************************
61:      * PREPROCESSOR DEFINITIONS *
62:      ****************************/
63:
64:     /*
65:         defines for DT-2871 board specifying the addresses of the I/O
66:         and memory bases and the register offsets from I/O base, modes
67:         and other associated information
68:     */
69: #define MEM_BASE    0x000A0000ul    /* base segment of board's memory space */
70: #define X_SIZE      512             /* size of one display line in pixels   */
71: #define Y_SIZE      512             /* number of display lines in screen    */
72: #define SCREEN_SIZE 0x00040000ul    /* size of one screen buffer in bytes   */
73:
74: #define IO_BASE     0x0230      /* base address of board's I/O space        */
75: #define     IN_CSR_1    0x00    /* video input control/status register 1    */
76: #define     IN_CSR_2    0x02    /* video input control/status register 2    */
77: #define     OUT_CSR     0x04    /* video output control/status register     */
78: #define     CURSOR      0x06    /* cursor register: cursor pixel and line   */
79: #define     INDEX       0x08    /* index register: LUT index                */
80: #define     X_PAN       0x08    /* X pan register: X direction pans         */
81: #define     IN_LUT      0x0A    /* input LUT entry register                 */
82: #define     R_LUT       0x0A    /* result LUT entry register                */
83: #define     Y_PAN       0x0A    /* Y pan register: Y direction pans         */
84: #define     RED_GREEN   0x0C    /* red/green output LUT entry register      */
85: #define     START       0x0C    /* start register: starting line and pixel  */
86: #define     BLUE_       0x0E    /* blue output LUT entry register           */
87: #define     END         0x0E    /* end register: ending line and pixel      */
88:
89: #define VIDEO_IN    0x0000      /* video input mode                         */
90:                                 /* 0x0010 is reserved mode                  */
91: #define LD_IN_LUT   0x0020      /* load input LUT mode                      */
92: #define LD_R_LUT    0x0030      /* load result LUT mode                     */
93:                                 /* 0x0040 is reserved mode                  */
94: #define SLOW_SCAN   0x0050      /* slow scan video input mode               */
95: #define PORT_OUT    0x0060      /* send data to output port mode            */
96: #define PORT_IN     0x0070      /* recieve data from input port mode        */
97:
98:
99:     /*** define macros to access extended memory ***/
100: #define MK_EP(segment, offset) ((doubleword)((unsigned long) \
101:             ((unsigned long)(segment) << 4) + (unsigned long)(offset)))
102: #define LOW_WORD(address) ((word)((unsigned long)(address)))
103: #define HIGH_BYTE(address) ((byte)((unsigned long)(address) >> 16))
104:
105:
106:     /*** defines for disk buffering and memory transfers ***/
107: #define LINES_PER_BLOCK     64      /* number of image lines in one block   */
108: #define DISK_BUF_SIZE       32767   /* size of disk buffer in bytes         */
109:
```

```
110:
112:     /*** define a macro to check if the board is BUSY ***/
113: #define DT_2871_BUSY (inport(IO_BASE+IN_CSR_1) & 0x0080)
114:
115:
116:
117:
118:     /************************
119:      * FUNCTION PROTOTYPES *
120:      ************************/
121:
122: void access_buffer(int buffer);
123: void stop_operations(void);
124:
125:
126:
127:     /************************
128:      * FUNCTION DEFINITIONS *
129:      ************************/
130:
131: /*-------------------------------------------------------------------------*/
132: void access_buffer(int buffer)
133:
134:     /*
135:           This routine programs the board so it has access to the
136:         given buffer in the memory space of the IBM PC AT.  It does
137:         this by first reading the OUT_CSR register, changing the bits
138:         necessary to allow access to the buffer, and writing the
139:         contents back to the OUT_CSR register.  Valid buffers are 0-15.
140:     */
141:
142:     {
143:
144:     outport(IO_BASE+OUT_CSR, (inport(IO_BASE+OUT_CSR) & 0xf0ff) |
145:                              ((buffer & 0x0f)<<8));
146:
147:     }   /* access_buffer */
148: /*-------------------------------------------------------------------------*/
149:
150:
151:
152: /*-------------------------------------------------------------------------*/
153: byte clear_screen(byte intensity, byte buffer)
154:
155:     /*      This module clears the given buffer to the given value
156:         by filling the memory for that screen with the given value.
157:         Note: this routine is affected by bit plane protection.  Also,
158:         the given buffer cannot be a buffer in use by another operation
159:         (such as being displayed)--nothing will happen as a result.  If
160:         the buffer is unable to be cleared, an error code is returned.
161:     */
162:
163:
164:     {
```

```
165:     doubleword  e_ptr;      /* pointer to extended memory to clear  */
167:     byte        val;        /* return value                         */
168:
169:         /*** calculate extended address of display buffer ***/
170:     e_ptr = MK_EP(MEM_BASE, (buffer&0x0001)*SCREEN_SIZE);
171:
172:
173:         /*** allow access to required buffer ***/
174:     access_buffer(buffer);
175:
176:
177:         /*** clear all lines on screen ***/
178:     val = memset_386(e_ptr, intensity, (doubleword)SCREEN_SIZE);
179:     _fpreset();
180:     return(val);
181:
182:     }   /* clear_screen */
183: /*-------------------------------------------------------------------------*/
184:
185:
186:
187: /*-------------------------------------------------------------------------*/
188: void display(byte state)
189:
190:     /*
191:             This routine is used to turn the display ON or OFF.  This is
192:         done by reading the current state of the OUT_CSR register, changing
193:         only the bit necessary to turn the display on or off, and then
194:         writing the result back to the OUT_CSR register.
195:     */
196:
197:
198:     {
199:
200:     if (state==OFF)
201:         outport(IO_BASE+OUT_CSR, (inport(IO_BASE+OUT_CSR) & 0xffdf));
202:     else
203:         outport(IO_BASE+OUT_CSR, (inport(IO_BASE+OUT_CSR) | 0x0020));
204:
205:     }   /* display */
206: /*-------------------------------------------------------------------------*/
207:
208:
209:
210: /*-------------------------------------------------------------------------*/
211  void display_buffer(byte buffer)
212:
213:     /*
214:             This routine programs the DT-2871 to display the given
215:         buffer.  Valid buffer numbers are 0-15.  To do this, it calls
216:         the routine to allow access to the required buffer, reads in
217:         the Y_PAN register (which holds the number of the output buffer),
218:         and changes only the appropriate bits to make the given buffer
219:         the current output buffer, and then writes the result back out to
```

```
220:         the Y_PAN register.  This routine will not work if used while the
222:         board is in any load LUT mode.
223:     */
224:
225:
226:     {
227:
228:     outport(IO_BASE+Y_PAN, (inport(IO_BASE+Y_PAN) & 0x0fff)
229:                          | ((buffer & 0x0f)<<12));
230:
231:     }   /* display_buffer */
232: /*--------------------------------------------------------------------*/
233:
234:
235:
236: /*--------------------------------------------------------------------*/
237: byte init_board(void)
238:
239:     /*
240:             This routine initializes the DT-2871 board by:
241:         1) initializing the OUT_CSR register to turn the display off
242:         2) checking to see if board is present
243:         3) initializing the IN_CSR_1 register to stop board operations
244:         4) programming the input look-up tables (they are made linear)
245:         5) programming the output look-up tables (they are made linear)
246:         6) initializing the IN_CSR_2 register to anything except the
247:            LD_IN_LUT or LD_R_LUT modes
248:         7) initializing the X_PAN register (no panning; no zooming)
249:         8) initializing the Y_PAN register (no panning; display buffer 0)
250:         9) initializing the memory to all zeroes => a clear display
251:        10) setting the OUT_CSR register to turn the display on
252:        11) returning a value indicating that no error occurred
253:     */
254:
255:
256:     {
257:     lut_type    lut;        /* holds LUT entries to initialize LUT's    */
258:     int         val;        /* general value                            */
259:
260:
261:         /*** initialize OUT_CSR ***/
262:     outport(IO_BASE+OUT_CSR, 0x0000);   /*  bit 15, 13, 12, 7: read only
263:                                             bit 8: not used
264:                                             bit 14: no interrupts
265:                                             bit 13: read only
266:                                             bits 11-9: use frames 0 and 1
267:                                             bit 6: begin operating on BUSY
268:                                             bit 5: display off
269:                                             bit 4: cursor off
270:                                             bit 3: use internal clock
271:                                             bits 2-0: use output LUT 0
272:                                         */
273:
274:
```

```
275:        /*** check if board is present ***/
277:    if (inport(IO_BASE+OUT_CSR) & 0x4f7f) {
278:        print_error("ERROR: DT-2871 not present");
279:        return(1);
280:        }
281:
282:
283:        /*** initialize the IN_CSR_1 register ***/
284:    outport(IO_BASE+IN_CSR_1, 0x0000);  /* bits 15-12: ALU does function=A
285:                                           bit 11: dedicated feedback
286:                                           bits 10-9: use result LUT 0
287:                                           bit 8: use LUTs for no carry
288:                                           bit 7: don't start operation
289:                                           bit 6: no interrupts
290:                                           bit 5: don't pass busy bit
291:                                           bit 4: no carry into ALU
292:                                           bit 3: do math function
293:                                           bits 2-0: use input LUT 0
294:                                        */
295:    while (DT_2871_BUSY)                /* and wait until board is ready    */
296:        ;
297:
298:
299:        /*** program input LUTs ***/
300:    for (val=0; val<IN_LUT_SIZE; lut.in_lut[val] = val++);  /* initialize   */
301:
302:    for (val=0; val<NUM_IN_LUTS; val++)                     /* write        */
303:        write_lut(INPUT_LUT, val, 0, IN_LUT_SIZE-1, &lut);
304:
305:
306:        /*** program output LUTs ***/
307:    for (val=0; val<OUT_LUT_SIZE; val++) {          /* initialize entries   */
308:        lut.out_lut[val].red_green = val+256*val;
309:       .lut.out_lut[val].blue      = val;
310:        }
311:
312:    for (val=0; val<NUM_OUT_LUTS; val++)            /* write entries        */
313:        write_lut(OUTPUT_LUT, val, 0, OUT_LUT_SIZE-1, &lut);
314:
315:
316:        /*** initialize the IN_CSR_2 register ***/
317:    outport(IO_BASE+IN_CSR_2, 0x0f8f);  /* bit 15: bit planes 4-7 enabled   */
318:                                        /* bit 14: bit planes 2-3 enabled   */
319:                                        /* bit 13: bit plane 1 enabled      */
320:                                        /* bit 12: bit plane 0 enabled      */
321:                                        /* bits 11-8: use frame buffer 15   */
322:                                        /* bit 7: frame is not zoomed, etc. */
323:                                        /* bits 6-4: video input mode       */
324:                                        /* bits 3-0: use frame buffer 15    */
325:
326:
327:        /*** initialize the X_PAN register ***/
328:    outport(IO_BASE+X_PAN, 0x0000);     /* no zooming or X panning          */
329:
```

```
330:
332:        /*** initialize the Y_PAN register ***/
333:    outport(IO_BASE+Y_PAN, 0x0000);     /* display buffer 0; no Y panning   */
334:
335:
336:        /*** initialize the OUT_CSR register ***/
337:    outport(IO_BASE+OUT_CSR, 0x0020);   /* as before except display on      */
338:
339:
340:        /*** return value indicating no error occurred ***/
341:    return(0);
342:
343:    }   /* init_board */
344: /*--------------------------------------------------------------------------*/
345:
346:
347:
348: /*--------------------------------------------------------------------------*/
349: void print_header_info(header_type header)
350:
351:    /*
352:            This module recieves an image header as an input
353:        parameter.  It then takes the information stored in the
354:        header and prints it in a human readable format using the
355:        conio routines so the color of the text it prints may be
356:        changed by the calling module.
357:    */
358:
359:
360:    {
361:    byte    comment[257];   /* comment in image */
362:
363:
364:    cprintf("%*cImage width:  %d\n\r", ENTRY_INDENT, ' ', (int)(header[4]+256*header[5]));
365:    cprintf("%*cImage height: %d\n\r", ENTRY_INDENT, ' ', (int)(header[6]+256*header[7]));
366:    cprintf("%*cCoordinates of original X-axis position: (%d,%d)\n\r",
367:            ENTRY_INDENT, ' ', (int)(header[8]), (int)(header[9]));
368:    cprintf("%*cCoordinates of original Y-axis position: (%d,%d)\n\r",
369:            ENTRY_INDENT, ' ',(int)(header[10]), (int)(header[11]));
370:    cprintf("%*cFile type: ", ENTRY_INDENT, ' ');
371:
372:    switch (header[12]+256*header[13]) {
373:        case 0: cprintf("NORMAL (can be correctly read by this program)\n\r");
374:                break;
375:        case 1: cprintf("COMPRESSED (must be uncompressed to be read correctly"
376:                        " by this program)\n\r");
377:                break;
378:        case 2: cprintf("SPECIAL (unspecifed type; must be converted to IMAGEACTION"
379:                        " format to be read correctly by this program\n\r");
380:                break;
381:        default:    cprintf("UNKNOWN (illegal file type)\n\r");
382:        }   /* switch */
383:
384:    memcpy(comment, header+IMAGE_HEADER_SIZE, (int)(header[2]+256*header[3]));
```

```
385:     comment[header[2]+256*header[3]+1] = '\0';
387:     cprintf("%*cComment:\n\r", ENTRY_INDENT, ' ');
388:     cprintf("%*c        %s\n\r", ENTRY_INDENT, ' ', comment);
389:
390:     }   /* print_header_info */
391: /*-------------------------------------------------------------------------*/
392:
393:
394:
395: /*-------------------------------------------------------------------------*/
396: byte read_image(byte buffer, byte quad, char *pathname, header_type header)
397:
398:     /*
399:           This module reads in an AOI (a quadrant of the screen (0-3) or
400:         the entire screen (quad=0).  The header from the image is returned.
401:         The image may be less than a full quadrant but it must be square.
402:         If an image is larger than a quadrant, it is assumed to be a full
403:         screen image.  The quadrant is with respect to the given buffer
404:         frame.
405:     */
406:
407:
408:     {
409:     doubleword  block_1;        /* extended address of line buffer  */
410:     doubleword  block_2;        /* extended address of image buffer */
411:     FILE        *file_ptr;      /* pointer for image file           */
412:     word        i;              /* general loop variable            */
413:     byte        *line_buffer;   /* image line buffer                */
414:     byte        val=0;          /* return value                     */
415:     word        x_size;         /* number of pixels in 1 image line */
416:     word        y_size;         /* number of lines in image         */
417:
418:
419:         /*** allocate memory for image line buffer ***/
420:     if ((line_buffer=malloc(LINES_PER_BLOCK*X_SIZE)) == NULL) {
421:         print_error("ERROR: Insufficient memory to allocate line buffer.\n");
422:         return(1);
423:         }
424:
425:
426:         /*** calculate extended address of line buffer ***/
427:     block_1 = MK_EP(FP_SEG((void huge *)line_buffer),
428:                     FP_OFF((void huge *)line_buffer));
429:
430:
431:         /*** calculate extended address of image buffer ***/
432:     block_2 = MK_EP(MEM_BASE, (buffer&0x0001)*SCREEN_SIZE);
433:
434:
435:         /*** open image file and buffer it ***/
436:     if ((file_ptr=fopen(pathname, "rb")) == NULL) {
437:         cprintf("Could not open %s\n\r", pathname);
438:         free(line_buffer);
439:         return(2);
```

```
440:         }
442:     if (setvbuf(file_ptr, NULL, _IOFBF, DISK_BUF_SIZE)) {
443:         cprintf("Could not buffer image.\n\r");
444:         fclose(file_ptr);
445:         free(line_buffer);
446:         return(3);
447:         }
448:
449:
450:         /*** read image header ***/
451:     fread(header, IMAGE_HEADER_SIZE, sizeof(byte), file_ptr);   /* read header          */
452:     x_size = header[2] + 256*header[3];                         /* get size of comment  */
453:     fread(header+IMAGE_HEADER_SIZE, x_size, sizeof(byte), file_ptr);  /* read comment   */
454:     x_size = header[4] + 256*header[5];                         /* get image X size     */
455:     y_size = header[6] + 256*header[7];                         /* get image Y size     */
456:
457:
458:         /*** check to make sure parameters are in range ***/
459:     if (x_size > X_SIZE) {
460:         printf("Image size in X is larger than screen size in X\n");
461:         fclose(file_ptr);
462:         free(line_buffer);
463:         return(4);
464:         }
465:     if (y_size > Y_SIZE) {
466:         printf("Image size in Y is larger than screen size in Y\n");
467:         fclose(file_ptr);
468:         free(line_buffer);
469:         return(4);
470:         }
471:     if (x_size > X_SIZE/2  ||  y_size > Y_SIZE/2)
472:         quad = 0;
473:
474:
475:         /*** calculate starting address of quad ***/
476:     if (quad>1)
477:         block_2 += (quad-2)*X_SIZE/2+SCREEN_SIZE/2;
478:     else
479:         block_2 += quad*X_SIZE/2;
480:
481:
482:         /*** allow access to required buffer ***/
483:     access_buffer(buffer);
484:
485:
486:         /*
487:            for each line in the image: read the line into the line buffer and
488:            then transfer the contents of line buffer to the extended memory
489:            buffer
490:         */
491:     for (i=0; i<y_size/LINES_PER_BLOCK; i++) {
492:         fread(line_buffer, x_size, LINES_PER_BLOCK, file_ptr);
493:         if ((val=image_move_386(block_1, block_2, x_size,
494:                                 LINES_PER_BLOCK, X_SIZE-x_size))>0)
```

```
            break;
        block_2 += (doubleword)LINES_PER_BLOCK*X_SIZE;
        }   /* for */

        /*** close file and free line buffer memory ***/
    fclose(file_ptr);
    free(line_buffer);
    if (val)
        return(6+val);          /* return error code or */
    else
        return(0);              /* return code indicating no error occurred */

    }   /* read_image */
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
byte read_2_images(byte buffer, char *pathname_1, header_type header_1,
                   byte quad,   char *pathname_2, header_type header_2)

    /*
          This module reads in two images into the two consecutive
        quadrants specified by the quad parameter (which may be 0 or 2)
        into the given buffer.  The images are assumed to be one quadrant
        wide and long.  The headers from the images are returned.
    */


    {
    doubleword  block_1;        /* extended address of line buffer  */
    doubleword  block_2;        /* extended address of image buffer */
    FILE        *file_ptr_1;    /* pointer for image file 1         */
    FILE        *file_ptr_2;    /* pointer for image file 2         */
    size_t      i;              /* general loop variable            */
    byte        *line_buffer;   /* image line buffer                */
    doubleword  ptr;            /* generic pointer                  */
    byte        val;            /* return value                     */
    word        x_size_1;       /* number of pixels in 1 image line */
    word        y_size_1;       /* number of lines in image         */
    word        x_size_2;       /* number of pixels in 1 image line */
    word        y_size_2;       /* number of lines in image 2       */


        /*** allocate memory for image line buffer ***/
    if ((line_buffer=malloc(LINES_PER_BLOCK*X_SIZE)) == NULL) {
        printf("Insufficient memory to allocate line buffer.\n");
        return(1);
        }

        /*** calculate extended address of line buffer ***/
    block_1 = MK_EP(FP_SEG((void huge *)line_buffer),
                    FP_OFF((void huge *)line_buffer));

```

```
550:
552:        /*** calculate extended address of image buffer ***/
553:    block_2 = MK_EP(MEM_BASE, (buffer&0x0001)*SCREEN_SIZE);
554:
555:
556:        /*** open image files ***/
557:    if ((file_ptr_1 = fopen(pathname_1, "rb"))==NULL) {
558:        printf("Could not open %s\n", pathname_1);
559:        free(line_buffer);
560:        return(2);
561:        }
562:    if (setvbuf(file_ptr_1, NULL, _IOFBF, DISK_BUF_SIZE)) {
563:        printf("Could not buffer image 1.\n");
564:        fclose(file_ptr_1);
565:        free(line_buffer);
566:        return(3);
567:        }
568:
569:    if ((file_ptr_2 = fopen(pathname_2, "rb"))==NULL) {
570:        printf("Could not open %s\n", pathname_2);
571:        fclose(file_ptr_1);
572:        free(line_buffer);
573:        return(4);
574:        }
575:    if (setvbuf(file_ptr_2, NULL, _IOFBF, DISK_BUF_SIZE)) {
576:        printf("Could not buffer image 2.\n");
577:        fclose(file_ptr_1);
578:        fclose(file_ptr_2);
579:        free(line_buffer);
580:        return(5);
581:        }
582:
583:
584:        /*** read image headers ***/
585:    /*
586:        For each image:
587:       1) read header info, 2) determine size of comment, 3) skip comment,
588:       4) determine size of image from header info
589:    */
590:    fread(header_1, IMAGE_HEADER_SIZE, sizeof(byte), file_ptr_1);
591:    x_size_1 = header_1[2] + 256*header_1[3];
592:    fread(header_1+IMAGE_HEADER_SIZE, x_size_1, sizeof(byte), file_ptr_1);
593:    x_size_1 = header_1[4] + 256*header_1[5];
594:    y_size_1 = header_1[6] + 256*header_1[7];
595:
596:    fread(header_2, IMAGE_HEADER_SIZE, sizeof(byte), file_ptr_2);
597:    x_size_2 = header_2[2] + 256*header_2[3];
598:    fread(header_2+IMAGE_HEADER_SIZE, x_size_2, sizeof(byte), file_ptr_2);
599:    x_size_2 = header_2[4] + 256*header_2[5];
600:    y_size_2 = header_2[6] + 256*header_2[7];
601:
602:
603:        /*** check to make sure parameters are in range ***/
604:    if ((x_size_1+x_size_2)>X_SIZE) {
```

```
605:          printf("Image size in X is larger than one quadrant.\n");
607:          fclose(file_ptr_1);
608:          fclose(file_ptr_2);
609:          free(line_buffer);
610:          return(6);
611:          }
612:      if ((y_size_1+y_size_2)>Y_SIZE) {
613:          printf("Image size in Y is larger than one quadrant.\n");
614:          fclose(file_ptr_1);
615:          fclose(file_ptr_2);
616:          free(line_buffer);
617:          return(6);
618:          }
619:
620:
621:          /*** calculate starting address of quad (quad=0 or 2) ***/
622:      if (quad==2)
623:          block_2 += SCREEN_SIZE/2;
624:
625:
626:          /*** allow access to required buffer ***/
627:      access_buffer(buffer);
628:
629:
630:          /*
631:             read each line of the images into the line buffer
632:             if the buffer is full (halfway through the image and at the end of
633:             the image), transfer the block to extended memory
634:             (y_size_2 is offset of current line in line buffer in conventional
635:             memory and offset is offset of current line in buffer in
636:             extended memory)
637:          */
638:
639:      ptr = block_2;
640:      for (i=0; i<y_size_1/LINES_PER_BLOCK; i++) {
641:          fread(line_buffer, x_size_1, LINES_PER_BLOCK, file_ptr_1);
642:          if ((val=image_move_386(block_1, ptr, x_size_1,
643:                                  LINES_PER_BLOCK, X_SIZE-x_size_1))>0) {
644:              fclose(file_ptr_1);
645:              fclose(file_ptr_2);
646:              free(line_buffer);
647:              return(6+val);
648:              }
649:          ptr += (doubleword)LINES_PER_BLOCK*X_SIZE;
650:          }
651:
652:      ptr = block_2 + X_SIZE/2;
653:      for (i=0; i<y_size_2/LINES_PER_BLOCK; i++) {
654:          fread(line_buffer, x_size_2, LINES_PER_BLOCK, file_ptr_2);
655:          if ((val=image_move_386(block_1, ptr, x_size_2,
656:                                  LINES_PER_BLOCK, X_SIZE-x_size_2))>0) {
657:              fclose(file_ptr_1);
658:              fclose(file_ptr_2);
659:              free(line_buffer);
```

```
660:            return(6+val);
662:            }
663:        ptr += (doubleword)LINES_PER_BLOCK*X_SIZE;
664:        }
665:
666:
667:        /*** close files and free line buffer memory ***/
668:    fclose(file_ptr_1);
669:    fclose(file_ptr_2);
670:    farfree(line_buffer);
671:    return(0);                  /* return code indicating no error occurred */
672:
673:    }   /* read_2_images */
674: /*-------------------------------------------------------------------------*/
675:
676:
677:
678: /*-------------------------------------------------------------------------*/
679: void screen_hold(word num_fields)
680:
681:    /*
682:            This routine will wait for the specified number of fields
683:        to pass before control is returned to the caller.  It does this
684:        by repeatedly waiting for the V_SYNC bit of OUT_CSR to go low
685:        (indicating the display is being updated) and then to go high
686:        (indicating the display is in a vertical retrace cycle).  It
687:        repeats the above checks, num_fields times.  This routine
688:        returns during the vertical retrace cycle of the last field.
689:        Note that if the routine is called with num_fields equal to
690:        zero, control is returned to the caller as soon as we are in
691:        the vertical retrace cycle.  This is provided so a sequence of
692:        timed screens can begin at the proper time, in the vertical
693:        retrace cycle, by syncing the sequence to the vertical retrace.
694:    */
695:
696:
697:    {
698:
699:    if (num_fields==0) {
700:        while (!(inport(IO_BASE+OUT_CSR) & 0x8000))
701:            ;                   /* wait while screen is being updated   */
702:        return;                 /* return when in retrace cycle */
703:        }
704:
705:    while (num_fields--) {
706:        while (inport(IO_BASE+OUT_CSR) & 0x8000)
707:            ;                   /* wait while in vertical retrace cycle */
708:        while (!(inport(IO_BASE+OUT_CSR) & 0x8000))
709:            ;                   /* wait while screen is being updated   */
710:        }   /* while */
711:
712:    }   /* screen_hold */
713: /*-------------------------------------------------------------------------*/
714:
```

```
715:
717:
718: /*-------------------------------------------------------------------------*/
719: void set_write_protect(byte value)
720:
721:     /*
722:           This routine allows the user to enable or disable write
723:         protection from any given bit plane.  A one disables a write
724:         to the given bit plane.  Bit 0 of value protects BP 0, bit
725:         1 protects BP 2 and 3, bit 3 protects BP 4-7.
726:     */
727:
728:
729:     {
730:
731:         /*** wait for end of operation ***/
732:     while (DT_2871_BUSY)
733:         ;
734:
735:         /*** set write protect bits ***/
736:     outport(IO_BASE+IN_CSR_2, (inport(IO_BASE+IN_CSR_2) & 0x0fff) |
737:           ((value & 0x0f) << 12));
738:
739:     }   /* set_write_protect */
740: /*-------------------------------------------------------------------------*/
741:
742:
743:
744: /*-------------------------------------------------------------------------*/
745: void stop_operations(void)
746:
747:     /*
748:           This routine stops all operations on the board.  It reads
749:         the IN_CSR_1 register, ANDs the PASS bit (5) to 0 which will
750:         cause the board to stop all operations at the end of the operation
751:         in progress, writes the result back to the IN_CSR_1 register, and
752:         waits for the BUSY bit (7) to clear which indicates that the
753:         operation which was in progress has completed.
754:     */
755:
756:
757:     {
758:
759:         /*** tell board to stop at end of current operation ***/
760:     outport(IO_BASE+IN_CSR_1, inport(IO_BASE+IN_CSR_1) & 0xffdf);
761:
762:
763:         /*** wait for end of current operation ***/
764:     while (DT_2871_BUSY)
765:         ;
766:
767:     }   /* stop_operations */
768: /*-------------------------------------------------------------------------*/
769:
```

```
770:
772:
773: /*-------------------------------------------------------------------------*/
774: void write_lut(byte type_of_lut, byte lut_number, word start, word stop,
775:                lut_type *lut)
776:
777:     /*
778:           This module initializes the values from start to stop in the
779:         given LUT number to the entries given in the LUT array.  This
780:         routine can initialize either an input, output, or result LUT.
781:         If the lut sent it is not of the type specified by type_of_lut,
782:         results are unpredictable.
783:     */
784:
785:
786:     {
787:     int in_csr_1;   /* old value of the input control/status register 1 */
788:     int in_csr_2;   /* old value of the input control/status register 2 */
789:     int out_csr;    /* old value of the output control/status register  */
790:     int val;        /* general value                                    */
791:
792:
793:         /*** make sure we have a valid lut type ***/
794:     if (type_of_lut!=INPUT_LUT && type_of_lut!=OUTPUT_LUT &&
795:         type_of_lut!=RESULT_LUT)
796:         return;
797:
798:
799:         /*** stop board operations ***/
800:     stop_operations();
801:
802:
803:         /*** read and save appropriate registers ***/
804:     in_csr_2 = inport(IO_BASE+IN_CSR_2);        /* save IN_CSR_2 for all    */
805:     if (type_of_lut != OUTPUT_LUT)              /* save IN_CSR_1 if LUT     */
806:         in_csr_1 = inport(IO_BASE+IN_CSR_1);    /* type is INPUT or RESULT  */
807:     else                                        /* else LUT type is OUTPUT  */
808:         out_csr  = inport(IO_BASE+OUT_CSR);     /* so save OUT_CSR          */
809:
810:
811:         /*** set the board to load appropriate LUT mode ***/
812:     if (type_of_lut != RESULT_LUT)              /* if LUT type is INPUT or  */
813:         outport(IO_BASE+IN_CSR_2, LD_IN_LUT);   /* OUPUT, set mode to load  */
814:     else                                        /* input LUT; else set mode */
815:         outport(IO_BASE+IN_CSR_2, LD_R_LUT);    /* to load result LUT       */
316:
817:
818:         /*** program each entry in the LUT ***/
819:     for (val=start; val<=stop; val++) {
820:
821:         /* select appropriate LUT; then select entry; then write entry */
822:         switch (type_of_lut) {
823:             case INPUT_LUT: outport(IO_BASE+IN_CSR_1,
824:                                     (in_csr_1 & 0xfff8) |   /* clear LUT bits   */

FILE=DT2871.C    Fri Jun 16 01:58:16 1989    PAGE=15
```

```
825:                                    (lut_number & 0x07) |   /* set LUT bits     */
827:                                    (val & 0x0100));        /* set high/low LUT */
828:                            outport(IO_BASE+INDEX, val & 0x00ff);
829:                            outport(IO_BASE+IN_LUT, lut->in_lut[val]);
830:                            break;
831:            case OUTPUT_LUT:outport(IO_BASE+OUT_CSR,
832:                                    (out_csr & 0xfff8) |    /* clear LUT bits   */
833:                                    (lut_number & 0x07));   /* set LUT bits     */
834:                            outport(IO_BASE+INDEX, val & 0x00ff);
835:                            outport(IO_BASE+RED_GREEN, lut->out_lut[val].red_green);
836:                            outport(IO_BASE+BLUE_, lut->out_lut[val].blue);
837:                            break;
838:            case RESULT_LUT:outport(IO_BASE+IN_CSR_1,
839:                                    (in_csr_1 & 0xf9ff) |   /* clear LUT bits   */
840:                                    ((lut_number & 0x03)<<9) | /* set LUT bits  */
841:                                    (val & 0x0100));        /* set high/low LUT */
842:                            outport(IO_BASE+INDEX, val & 0x00ff);
843:                            outport(IO_BASE+IN_LUT, lut->result_lut[val]);
844:            } /* switch */
845:
846:        } /* for */
847:
848:        /*** restore appropriate registers ***/
849:    outport(IO_BASE+IN_CSR_2, in_csr_2);        /* always restore IN_CSR_2  */
850:    if (type_of_lut != OUTPUT_LUT)              /* restore IN_CSR_1 if type */
851:        outport(IO_BASE+IN_CSR_1, in_csr_1);    /* is RESULT or INPUT; else */
852:    else                                        /* restore OUT_CSR          */
853:        outport(IO_BASE+OUT_CSR, out_csr);
854:
855:    } /* write_lut */
```

```
 1: /*****************************************************************************
 2:
 3:         FILENAME:               386move.h
 4:         PROGRAMMER:             Christopher Voltz - UDRI
 5:         CREATED:                -1/8811.21
 6:         LAST MODIFIED:          -1/8801.06
 7:         INTERFACE PROTOCOL:     Turbo C 2.0 / TASM 1.0
 8:         REQUIREMENTS:           386 processor.
 9:
10:
11:         This file provides the types and function prototypes necessary to
12:     implement a routine which transfers a block of memory from the real mode
13:     address space to the protected mode address space (extended memory).
14:
15: *****************************************************************************/
16:
17:
18:     /*** check to ensure required types are declared ***/
19: #ifndef _BYTE
20: #define _BYTE 1
21: typedef unsigned char       byte;   /* define a byte (8-bit) type */
22: #endif
23:
24: #ifndef _WORD
25: #define _WORD 1
26: typedef unsigned short int  word;   /* define a word (16-bit) type */
27: #endif
28:
29: #ifndef _DWORD
30: #define _DWORD 1
31: typedef unsigned long int   doubleword; /* define a doubleword (32-bit) typ
32: #endif
33:
34:
35:     /*** check to ensure proper memory model ***/
36: #ifndef __LARGE__
37: #error Memory model size must be large.
38: #endif
39:
40:
41:     /*** function prototypes ***/
42: byte image_move_386(doubleword from, doubleword to, word width,
43:                     word length, word skip);
44: byte memmove_386(doubleword from, doubleword to, doubleword num_dwords);
45: byte memset_386(doubleword ptr, byte val, doubleword num_bytes);
```

```
                                ;************************
                                ;*  ASSEMBLER OPTIONS   *
                                ;************************

IDEAL                   ;TASM ideal mode
%TABSIZE    4           ;set tabsize in listing file to 4 spaces
%PAGESIZE   66, 132     ;wide listing, normal height
WARN                    ;full warnings generation except disable errors
NOWARN      PRO         ;relating to CS overide in protected mode
MODEL       LARGE, C    ;large model for TURBO C
P386                    ;386 protected mode instructions enabled


%TITLE  "80386 extended memory move routines"
%SUBTTL "PROGRAM HEADER"
%NEWPAGE

; ********************************************************************************
; *                                                                              *
; *         FILENAME:           386move.asm                                      *
; *         CREATED:            -1/8811.11                                       *
; *         LAST MODIFIED:      -1/8901.11                                       *
; *         PROGRAMMER:         Christopher Voltz                                *
; *         INTERFACE PROTOCOL: TURBOC 2.0 / TASM 1.0                            *
; *                                                                              *
; *     These modules provide the code necessary to read a file from disk        *
; * and place it into extended memory.                                           *
; *                                                                              *
; ********************************************************************************

%SUBTTL "EQUATES"
%NEWPAGE

                                ;*************
                                ;*  EQUATES  *
                                ;*************

max_seg         EQU 0ffffH      ;maximum segment length in bytes
code_rights     EQU 10011000B   ;access rights byte for code (E/R, memory
                                ; segment, privilege 0, present)
data_rights     EQU 10010010B   ;access rights byte for data (R/W, memory
                                ; segment, privilege 0, present)
granularity_4k  EQU 10001111B   ;granularity byte for 4k granularity with 16-
                                ; bit addressing; segment limit = 0fxxxxH
granularity_1b  EQU 00000000B   ;granularity byte for 1 byte granularity with
                                ; 16-bit addressing; segment limit = 00xxxxH
stack_rights    EQU 10010110B   ;access rights byte for stack (R/W, memory
                                ; segment, privilege 0, present, expand down)
trap_gate       EQU 10000111B   ;access rights byte for a 286 trap gate,
                                ; present, privilege level 0, 0 copy count

cmos_control    EQU 070H        ;control port to write to display NMIs
cmos_data       EQU 071H        ;

FILE=386MOVE.ASM    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
  55: disable_nmi     EQU 10001111B   ;value to write to disable NMIs
  57: enable_nmi      EQU 00000000B   ;value to write to enable NMIs
  58:
  59: enable_virtual  EQU 000000001H  ;value to OR into CR0 to enter protected mode
  60: disable_virtual EQU 0fffffffeH  ;value to mask with CR0 to leave protected mode
  61:
  62: diagnostic_port EQU 061H        ;diagnostic port
  63: dos_interrupt   EQU 021H        ;interrupt level for DOS functions
  64: error_exit      EQU 04c02H      ;DOS function code to terminate program and
  65:                                 ;exit with error code = 2
  66: mfg_port        EQU 080H        ;manafacture's test port (used to hold value)
  67: parity_error    EQU 0c0H        ;port which contains parity error status
  68: ram_parity_off  EQU 00cH        ;value to write to turn parity checking off
  69: ram_parity_on   EQU 0f3H        ;value to write to turn parity checking on
  70:
  71: buffer_full     EQU 002H        ;indicates 8042's buffer is full
  72: control_port    EQU 060H        ;8042's control port
  73: disable_bit_20  EQU 0ddH        ;value to write to 8042 to disable A20 line
  74: enable_bit_20   EQU 0dfH        ;value to write to 8042 to enable A20 line
  75: status_port     EQU 064H        ;8042's status port
  76: write_output    EQU 0d1H        ;8042 code to allow write-thru to A20 gate
  77:
  78: high_nibble     EQU 0f0H        ;mask to get high nibble of byte
  79: low_nibble      EQU 00fH        ;mask to get low nibble of byte
  80: next_nibble     EQU 004H        ;number of bits to shift for next nibble
  81:
  82:
  83:
  84: %SUBTTL "STRUCTURE DEFINITIONS"
  85: %NEWPAGE
  86:
  87: ;                           ****************************
  88: ;                           *  STRUCTURE DEFINITIONS   *
  89: ;                           ****************************
  90:
  91: STRUC descriptor                    ;80386 segment descriptor
  92:     seg_limit       DW  0           ;segment limit
  93:     base_0_7        DB  0           ;base address bits 0..7
  94:     base_8_23       DW  0           ;base address bits 8..23
  95:     access_rights   DB  0           ;access rights for segment
  96:     granularity     DB  0           ;granularity
  97:     base_24_31      DB  0           ;base address bits 24..31
  98: ENDS descriptor
  99:
 100:
 101:
 102: ;----- open code segment and reset assembler assumptions
 103:     CODESEG USE16
 104:     ASSUME  CS:@code, DS:NOTHING, ES:NOTHING, FS:NOTHING, GS:NOTHING
 105:
 106:
 107:
 108: %SUBTTL "memmove_386"
 109: %NEWPAGE
```

```
110:
112: ; ************************************************************************
113: ; *                                                                      *
114: ; * byte memmove_386(doubleword from, doubleword to,                     *
115: ; *                  doubleword num_dwords);                             *
116: ; *     /*                                                               *
117: ; *        define a procedure to transfer a block of memory (which has   *
118: ; *        num_dwords doublewords in it) from one area (given by from)   *
119: ; *        to another area (given by to).  Either or both areas may be   *
120: ; *        in the extended memory space.                                 *
121: ; *     */                                                               *
122: ; *                                                                      *
123: ; ************************************************************************
124:
125: PROC    memmove_386 FAR
126:     GLOBAL  memmove_386:PROC
127:     ARG     from:DWORD, to:DWORD, num_dwords:DWORD
128:
129:
130: ;----- TASM automatically inserts entry code
131: ;   ENTER   0, 0
132:
133: ;----- enter protected mode
134:     CALL    NEAR enter_protected_mode
135:
136: ;----- get a pointer to source block in ESI; get a pointer to the destination
137: ;      block in EDI; and the count of the number of words to move in ECX
138:     MOV     ESI, [DWORD PTR SS:from]
139:     MOV     EDI, [DWORD PTR SS:to]
140:     MOV     ECX, [DWORD PTR SS:num_dwords]
141:
142: ;----- move the block of memory in double words
143:     REP     MOVS [WORD PTR ES:EDI], [WORD PTR DS:ESI]
144:
145: ;----- leave protected mode
146:     CALL    NEAR leave_protected_mode
147:
148: ;----- TASM automatically inserts exit code
149: ;   LEAVE
150:
151:     RET
152:
153: ENDP    memmove_386
154:
155:
156:
157: %SUBTTL "image_move_386"
158: %NEWPAGE
159:
160: ; ************************************************************************
161: ; *                                                                      *
162: ; * byte image_move_386(doubleword from, doubleword to, word width,     *
163: ; *                     word length, word skip);                         *
164: ; *     /*                                                               *

FILE=386MOVE.ASM    Fri Jun 16 01:58:16 1989    PAGE 3
```

```
165: ; *        define a procedure to move an image which is width bytes     *
167: ; *        wide and length bytes long.  When the image is copied        *
168: ; *        copied, however, after a line has been copied and the offset *
169: ; *        of the destination has been updated (by adding width to it), *
170: ; *        skip is also added to it to facilitate copying of block      *
171: ; *        images onto a memory mapped screen.  The source address is   *
172: ; *        given by from and the destination address is given by to.    *
173: ; *        Either or both areas may be in the extended memory space.    *
174: ; *     */                                                              *
175: ; *                                                                     *
176: ; **********************************************************************
177:
178: PROC    image_move_386 FAR
179:     GLOBAL  image_move_386:PROC
180:     ARG     from:DWORD, to:DWORD, line_width:WORD, num_lines:WORD, skip:WORD
181:
182:
183: ;----- TASM automatically inserts entry code
184: ;   ENTER   0, 0
185:
186: ;----- enter protected mode
187:     CALL    NEAR enter_protected_mode
188:
189: ;----- get a pointer to source block in ESI; get a pointer to the destination
190: ;      block in EDI; get the number of lines to move in EBX; get the width of
191: ;      the line in EDX; and get the num to skip in EAX
192:     MOV     ESI, [DWORD PTR SS:from]
193:     MOV     EDI, [DWORD PTR SS:to]
194:     MOVZX   EDX, [WORD PTR SS:line_width]
195:     MOVZX   EBX, [WORD PTR SS:num_lines]
196:     MOVZX   EAX, [WORD PTR SS:skip]
197:
198: ;----- account for movement of words rather than bytes
199:     SHR     EDX, 1
200:
201: ;----- move the block of memory
202: @@l1:
203:     MOV     ECX, EDX
204:     REP     MOVS [WORD PTR ES:EDI], [WORD PTR DS:ESI]
205:     ADD     EDI, EAX
206:     DEC     EBX
207:     JNE     SHORT @@l1
208:
209: ;----- leave protected mode
210:     CALL    NEAR leave_protected_mode
211:
212: ;----- TASM automatically inserts exit code
213: ;   LEAVE
214:
215:     RET
216:
217: ENDP    image_move_386
218:
219:
```

```
220:
222: %SUBTTL "memset_386"
223: %NEWPAGE
224:
225: ; ************************************************************************
226: ; *                                                                      *
227: ; * byte memset_386(doubleword ptr, byte val, doubleword num_bytes);     *
228: ; *                                                                      *
229: ; *     This routine initializes num_bytes bytes of memory pointed to    *
230: ; * by ptr to the given value (val).  The memory can be extended         *
231: ; * memory.                                                              *
232: ; *                                                                      *
233: ; ************************************************************************
234:
235: PROC    memset_386 FAR
236:     GLOBAL  memset_386:PROC
237:     ARG     mem_ptr:DWORD, val:BYTE, num_bytes:DWORD
238:
239:
240: ;----- TASM automatically inserts entry code
241: ;   ENTER   0, 0
242:
243: ;----- enter protected mode
244:     CALL    NEAR enter_protected_mode
245:
246: ;----- get a pointer to the block to be initialized in EDI; get the value to
247: ;      initialize the block to in AL; and get the count of the number of bytes
248: ;      to initialize in ECX
249:     MOV     EDI, [DWORD PTR SS:mem_ptr]
250:     MOV     AL, [BYTE PTR SS:val]
251:     MOV     ECX, [DWORD PTR SS:num_bytes]
252:
253: ;----- initialize the block of memory
254:     CLD                             ;set direction forward
255:     REP     STOS [BYTE PTR ES:EDI]  ;initialize the byte block
256:
257: ;----- leave protected mode
258:     CALL    NEAR leave_protected_mode
259:
260: ;----- TASM automatically inserts exit code
261: ;   LEAVE
262:
263:     RET
264:
265: ENDP    memset_386
266:
267:
268:
269: %SUBTTL "enter_protected_mode"
270: %NEWPAGE
271:
272: ; ************************************************************************
273: ; *                                                                      *
274: ; * PROC enter_protected_mode NEAR                                       *

FILE=386MOVE.ASM    Fri Jun 16 01:58:16 1989    PAGE=5
```

```
275: ; *                                                                    *
277: ; *     This routine places the 80386 into protected mode and returns  *
278: ; * to the caller.  This is done by disabling interrupts (including    *
279: ; * NMIs), initializing the GDTR and IDTR to point to tables in RAM    *
280: ; * created by this procedure, and switching to protected mode.  This  *
281: ; * routine leaves the DS and ES registers with the selector for a 4G  *
282: ; * data segment which is readable and writable.  The CS register holds *
283: ; * the selector for this code segment which is readable and          *
284: ; * executable.  Paging is not enabled.  Note: all 16-bit registers are *
285: ; * saved and left on the stack, so the calling routine's parameters   *
286: ; * will be pushed down by two quadwords.                             *
287: ; *                                                                    *
288: ; **********************************************************************
289:
290: LABEL   enter_protected_mode NEAR
291:
292:
293: ;----- save caller's state
294:     POP     AX                              ;save return address
295:     PUSH    DS                              ;save segment registers
296:     PUSH    ES
297:     PUSHA                                   ;save all 16-bit registers
298:     PUSH    AX                              ;restore return address
299:     MOV     [WORD PTR CS:old_ss], SS        ;save stack pointer
300:     SIDT    [PWORD PTR CS:old_idt]          ;save idt pointer.
301:     SGDT    [PWORD PTR CS:old_gdt]          ;save gdt pointer
302:
303: ;----- clear error flag
304:     SUB     AL, AL
305:     OUT     mfg_port, AL
306:
307: ;----- make a 32-bit address out of CS:idt_start
308:     MOV     AX, OFFSET idt_start
309:     MOVZX   EAX, AX
310:     MOV     BX, CS
311:     MOVZX   EBX, BX
312:     SHL     EBX, next_nibble
313:     ADD     EAX, EBX
314:
315: ;----- setup interrupt descriptor table base offset
316:     MOV     [DWORD PTR CS:idt_location], EAX
317:
318: ;----- make a 32-bit address out of CS:blockmove_gdt
319:     MOV     AX, OFFSET blockmove_gdt
320:     MOVZX   EAX, AX
321:     MOV     BX, CS
322:     MOVZX   EBX, BX
323:     SHL     EBX, next_nibble
324:     ADD     EAX, EBX
325:
326: ;----  setup global descriptor table base address
327:     MOV     [DWORD PTR CS:gdt_location], EAX
328:
329: ;----- setup stack segment base address
```

```
330:     MOV     AX, SS
332:     MOVZX   EAX, AX
333:     SHL     EAX, next_nibble
334:     MOV     [(descriptor PTR CS:flat_stack).base_0_7], AL
335:     SHR     EAX, 2*next_nibble
336:     MOV     [(descriptor PTR CS:flat_stack).base_8_23], AX
337:
338: ;----- setup code segment base address
339:     MOV     AX, CS
340:     MOVZX   EAX, AX
341:     SHL     EAX, next_nibble
342:     MOV     [(descriptor PTR CS:flat_code).base_0_7], AL
343:     SHR     EAX, 2*next_nibble
344:     MOV     [(descriptor PTR CS:flat_code).base_8_23], AX
345:
346: ;----- turn off interrupts and set direction flag
347:     CLI
348:     CLD
349:
350: ;----- disable NMIs
351:     MOV     AL, disable_nmi
352:     OUT     cmos_control, AL
353:
354: ;----- load new IDT and GDT values
355:     LIDT    [PWORD PTR CS:idt_descriptor]
356:     LGDT    [PWORD PTR CS:gdt_descriptor]
357:
358: ;----- gate address bit 20 on and exit if an error occurs
359:     SUB     AL, AL                  ;clear error code
360:     OUT     mfg_port, AL
361:     MOV     AH, enable_bit_20       ;enable address line 20 and above
362:     CALL    NEAR gate_a20
363:
364:     CMP     AL, 00                  ;if an error occurred then
365:     JE      SHORT @@l2              ;set the error flag and exit
366:     MOV     AL, 01                  ;othwerwise, continue at next block
367:     OUT     mfg_port, AL
368:     JMP     SHORT l6
369: @@l2:
370:
371: ;----- switch to virtual mode and purge prefetch queue and setup stack pointer
372:     MOV     EAX, CR0
373:     OR      EAX, enable_virtual
374:     MOV     CR0, EAX
375:     JMP     SHORT @@flush_1
376: @@flush_1:
377:     MOVZX   ESP, SP
378:
379: ;----- setup stack and data selectors
380:     MOV     AX, flat_stack-blockmove_gdt
381:     MOV     SS, AX                          ;setup stack selector
382:     MOV     AX, flat_data-blockmove_gdt
383:     MOV     DS, AX                          ;setup data segment selector
384:     MOV     ES, AX                          ;setup extra segment selector

FILE=386MOVE.ASM    Fri Jun 16 01:58:16 1989    PAGE=7
```

```
:35:     MOV     FS, AX                          ;setup the rest of the segment
387:     MOV     GS, AX                          ;regs to avoid illegal selector errors
388:
389: ;----- return to caller
390:     RET
391:
392:
393:
394: %SUBTTL "leave_protected_mode"
395: %NEWPAGE
396:
397: ; ********************************************************************************
398: ; *                                                                              *
399: ; * PROC leave_protected_mode NEAR                                               *
400: ; *                                                                              *
401: ; *     This routine returns the 80386 to real mode.  It enables all            *
402: ; * interrupts, restores the machine's original state (including all             *
403: ; * register values), sets the return code in AH and returns to the              *
404: ; * caller.                                                                      *
405: ; *     Exit codes are as follows:                                               *
406: ; * 1 => couldn't enable address line 20 and above                               *
407: ; * 2 => parity error                                                            *
408: ; * 3 => couldn't disable address line 20 and above                              *
409: ; *                                                                              *
410: ; ********************************************************************************
411:
412: LABEL   leave_protected_mode NEAR
413:
414:
415: ;----- if a parity error has occurred then clear the error and set the
416: ;      error code; in either case, return to real mode
417:     IN      AL, diagnostic_port
418:     AND     AL, parity_error
419:     JZ      SHORT real_mode
420:     MOV     AL, 02
421:     OUT     mfg_port, AL
422:
423: ;----- clear parity error
424:     MOV     AX, [WORD PTR ES:SI]    ;reset parity for source segment
425:     MOV     [WORD PTR ES:SI], AX
426:     MOV     AX, [WORD PTR DI]       ;reset parity for target segment
427:     MOV     [WORD PTR DI], AX
428:
429:     IN      AL, diagnostic_port
430:     JMP     SHORT @@l3              ;(delay)
431: @@l3:
432:     OR      AL, ram_parity_off
433:     OUT     diagnostic_port, AL
434:     JMP     SHORT @@l4              ;(delay)
435: @@l4:
436:     AND     AL, ram_parity_on
437:     OUT     diagnostic_port, AL
438:
439: ;----- return to real mode
```

```
real_mode:
    MOV     EAX, CR0                ;switch back to real mode
    AND     EAX, disable_virtual
    MOV     CR0, EAX
    JMP     SHORT @@flush_2         ;purge prefetch que
@@flush_2:

l6:
    MOV     AH, disable_bit_20      ;disable address line 20 and above
    CALL    NEAR gate_a20           ;if there is an error then check
    CMP     AL, 00                  ;if we already had an error
    JE      SHORT @@l7              ;if we did not, set the error code
    IN      AL, mfg_port            ;in either case, continue with exit code
    CMP     AL, 00
    JNE     SHORT @@l7
    MOV     AL, 03
    OUT     mfg_port, AL
@@l7:

;----- restore machine state
    POP     FS                      ;save return address
    LGDT    [PWORD PTR CS:old_gdt]
    LIDT    [PWORD PTR CS:old_idt]
    MOV     SS, [CS:old_ss]
    POPA
    POP     ES
    POP     DS
    PUSH    FS                      ;restore return address

;----- set return code; enable interrupts, and return
    MOV     AL, enable_nmi          ;enable NMIs
    OUT     cmos_control, AL

    IN      AL, mfg_port            ;set return code (in AL)
    MOV     AH, 0

    STI
    RET


%SUBTTL "gate_a20"
%NEWPAGE

; ************************************************************************
; *                                                                      *
; * PROC gate_a20 NEAR                                                   *
; *     /*                                                               *
; *         define a procedure to gate address line 20 on and off        *
; *     */                                                               *
; *                                                                      *
; ************************************************************************

PROC    gate_a20 NEAR
```

```
495:
497:     CALL    NEAR empty_8042
498:     JNZ     SHORT gate_a20_return
499:     MOV     AL, write_output
500:     OUT     status_port, AL
501:     CALL    NEAR empty_8042
502:     JNZ     SHORT gate_a20_return
503:     MOV     AL, AH
504:     OUT     control_port, AL
505:     CALL    empty_8042
506: gate_a20_return:
507:     RET
508: ENDP    gate_a20
509:
510:
511:
512: %SUBTTL "empty_8042"
513: %NEWPAGE
514:
515: ; ********************************************************************************
516: ; *                                                                              *
517: ; * PROC empty_8042 NEAR                                                         *
518: ; *      USES CX                                                                 *
519: ; *                                                                              *
520: ; *      /*                                                                      *
521: ; *          define a procedure to wait for the 8042's buffer to empty           *
522: ; *      */                                                                      *
523: ; *                                                                              *
524: ; ********************************************************************************
525:
526: PROC    empty_8042 NEAR
527:     USES CX
528:
529:     SUB     CX, CX
530: @@l1:
531:     IN      AL, status_port
532:     AND     AL, buffer_full
533:     LOOPNZ  @@l1
534:     RET
535: ENDP    empty_8042
536:
537:
538:
539: %SUBTTL "exception_handler"
540: %NEWPAGE
541:
542: ; ********************************************************************************
543: ; *                                                                              *
544: ; * exception_handler NEAR                                                       *
545: ; *                                                                              *
546: ; *      /*                                                                      *
547: ; *          define a routine to handle exceptions while in protected            *
548: ; *         mode                                                                 *
549: ; *      */                                                                      *
```

```
550: ; *                                                                    *
552: ; ******************************************************************************
553:
554: exception_handler:
555:     MOV     AX, flat_data-null          ;setup proper selector
556:     MOV     DS, AX
557:     MOV     EBX, 0b8000h+(2*80)*4       ;print error code
558:     ADD     AX, '0'
559:     MOV     [BYTE PTR DS:EDI], AL
560:     CALL    NEAR leave_protected_mode   ;return to real mode
561:     MOV     AX, error_exit              ;set error code and terminate program
562:     INT     dos_interrupt               ;using DOS interrupt function call
563:
564:
565:
566: %SUBTTL "DATA STORAGE AREA"
567: %NEWPAGE
568:
569: ;                       ***********************
570: ;                       *  DATA STORAGE AREA  *
571: ;                       ***********************
572:
573: LABEL   blockmove_gdt   BYTE        ;global descriptor table for block move
574:
575:     null            descriptor  <>  ;required dummy descriptor
576:
577:                                     ; 64K code segment based at CS:0
578:     flat_code       descriptor  <max_seg, 0, ?, code_rights, granularity_1b, 0>
579:
580:                                     ; 4-Gigabyte data segment based at 0
581:     flat_data       descriptor  <max_seg, 0, 0, data_rights, granularity_4k, 0>
582:
583:                                     ; 64K stack segment (expand down) at SS:0
584:     flat_stack      descriptor  <0, 0, ?, stack_rights, granularity_1b, 0>
585:
586: LABEL   end_blockmove_gdt   BYTE
587:
588:
589: ;----- define storage for processor state info
590: old_idt     DP      ?   ;old IDT info
591: old_gdt     DP      ?   ;old GDT info
592: old_ss      DW      ?   ;old stack segment register (SS)
593:
594: ;----- define storage for the code segment pseudodescriptor
595: gdt_descriptor:
596:                 DW      end_blockmove_gdt-blockmove_gdt-1   ;segment limit
597: gdt_location:   DD      ?                                   ;base address
598:
599: ;----- define storage for the IDT pseudodescriptor
600: idt_descriptor:
601:                 DW      idt_end-idt_start-1                 ;segment limit
602: idt_location    DD      ?                                   ;base address
603:
604:
```

```
605: %SUBTTL "INTERRUPT DESCRIPTOR TABLE"
607: %NEWPAGE
608:
609: ;                       *********************************
610: ;                       *  INTERRUPT DESCRIPTOR TABLE   *
611: ;                       *********************************
612:
613: ;----- define interrupt descriptor table
614: count = 0
615: idt_start:
616:     REPT    32
617:                                                 ;;offsets of handlers
618:         DW  OFFSET exceptions + count*(exceptions_end-exceptions)/32
619:         DW  flat_code-null                      ;;selectors of handlers => in
620:         DB  0                                   ;; GDT with privilege level 0
621:         DB  trap_gate                           ;;access byte for handlers
622:         DW  0
623:         count = count + 1
624:     ENDM
625: idt_end:
626:
627: count = 0
628: exceptions:
629:     REPT    32
630:         MOV     AX, count                       ;;load error code
631:         JMP     NEAR exception_handler          ;;goto real handler
632:         count = count + 1
633:     ENDM
634: exceptions_end:
635:
636:
637:
638: %SUBTTL "PRINT_VAL"
639: %NEWPAGE
640:
641: PROC     print_val NEAR
642:
643: ;   This routine prints the value in EAX by storing the appropriate ASCII
644: ;   codes at the location pointed to by EBX.  EBX is updated by two for each
645: ;   character printed to account for the attribute byte.  Two spaces are also
646: ;   printed following the 8 digit hexadecimal value in EAX.  This routine
647: ;   operates only in protected mode and assumes that a valid stack exists and
648: ;   that a data segment selector which points to a 4G segment at 0 is loaded
649: ;   into DS.
650:
651:     push    edx
652:     push    ecx
653:
654:     mov     ecx, 8
655:
656: @@1:
657:     shld    edx, eax, next_nibble
658:     shl     eax, next_nibble
659:     and     dl, 0fh

FILE=386MOVE.ASM    Fri Jun 16 01:58:16 1989    PAGE=12
```

```
650:     cmp     dl, 10
662:     jl      short @@2
663:     add     dl, 'A'-'9'-1
664: @@2:
665:     add     dl, '0'
666:     mov     [byte ptr ds:ebx], dl
667:     add     ebx, 2
668:     loopnz  short @@1
669:
670:     mov     ecx, 2
671:
672: @@3:
673:     mov     [byte ptr ds:ebx], ' '
674:     add     ebx, 2
675:     loopnz  short @@3
676:
677:     pop     ecx
678:     pop     edx
679:
680:     ret
681:
682: ENDP    print_val
683:
684: END
```

```
/*****************************************************************************

        FILENAME:               pcvision.h
        PROGRAMMER:             Christopher Voltz - UDRI
        CREATED:                -1/8802.04
        LAST MODIFIED:          -1/8904.18
        INTERFACE PROTOCOL:     Turbo C 1.5
        USAGE:                  program.c:
                                    #include <pcvision.h>

                                constant.h:
                                    #define _BYTE 1
                                    typedef unsigned char byte
                                    #endif
                                    #define ENTRY_INDENT 5
                                    #define ERROR_COLOR LIGHTRED+BLINK
                                    #define MENU_COLOR  WHITE


    This module provides the routines necessary to use the PCVision board.
Specifically, it provides: a clear routine to clear the screen to a given
intensity; an initialization routine to initialize the registers and LUTs on
the board; a LUT selection routine which sets the active LUT from a given
LUT set; a LUT write routine which allows the values of the active LUT to be
set; a routine to read in an AOI (or the entire screen) to be read from
the disk; a routine to hold a given screen for a specified number of
vertical blanking intervals; and a routine to display the contents of the
image header.
    This module is only the headers to the actual routines contained in
PCVISION.C

*****************************************************************************/



/************************
 * CONSTANT DEFINITIONS *
 ************************/

#define REG_BASE 0xFF40             /* register base (i/o) mapped           */
#define MEM_BASE 0xD000             /* memory base (segment)                */
#define CON_L REG_BASE+0x00         /* control register address (low)       */
#define CON_H REG_BASE+0x01         /* control register address (high)      */
#define LUT_ADDR REG_BASE+0x02      /* LUT address register address         */
#define LUT_DATA REG_BASE+0x03      /* LUT data register address            */
#define MASK REG_BASE+0x04          /* MASK register address                */
#define BLOCK_SELECT REG_BASE+0x05  /* block select register address        */
#define VERT_BLANK REG_BASE+0x06    /* vertical interrupt register address  */

    /** note: red and green were switched because the program was setup to
        use the greeen driver which malfunctioned; when it was decided to
        use the red driver instead it was simpler to make this change than
        to change the entire program.
    **/
```

```
#define RED_TABLE   1               /* Red LUT table   -- 0 */
#define GREEN_TABLE 0               /* Green LUT table -- 1 */
#define BLUE_TABLE  2               /* Blue LUT table  -- 2 */
#define INPUT_TABLE 3               /* Input LUT table -- 3 */

#define LUT_SIZE 256                /* size of LUT */


/*********************
 * TYPE DEFINITIONS *
 *********************/

#ifndef _BYTE                       /* define a byte type              */
#define _BYTE 1
typedef unsigned char byte
#endif
typedef byte lut_type[LUT_SIZE];    /* define a lut type               */
typedef byte header_type[320];      /* define an image file header type */
typedef byte quadrant[0xFFFF];      /* define memory for one quadrant   */


/************************
 * FUNCTION PROTOTYPES *
 ************************/

void Clear_screen(byte intensity);
void Freeze_mode(void);
void Grab_mode(void);
void Init_board(void);
void Print_header_info(header_type header);
void Read_image(byte quad, char *pathname, header_type header);
void Save_image(byte quad, char *pathname);
void Screen_hold(int num_screens);
void Set_lut(byte lut, byte lut_set);
void Write_lut(byte start, byte stop, lut_type lut);
```

```
/*****************************************************************************

        FILENAME:               pcvision.c
        PROGRAMMER:             Christopher Voltz - UDRI
        CREATED:                -1/8802.04
        LAST MODIFIED:          -1/8904.20
        INTERFACE PROTOCOL:     Turbo C 1.5
        USAGE:              program.c:
                                #include <pcvision.h>

                            constant.h:
                                #define _BYTE 1
                                typedef unsigned char byte
                                #endif
                                #define ENTRY_INDENT 5
                                #define ERROR_COLOR LIGHTRED+BLINK
                                #define FULL 4
                                #define MENU_COLOR  WHITE


    This module provides the routines necessary to use the PCVision board.
Specifically, it provides: a clear routine to clear the screen to a given
intensity; an initialization routine to initialize the registers and LUTs on
the board; a LUT selection routine which sets the active LUT from a given
LUT set; a LUT write routine which allows the values of the active LUT to be
set; a routine to read in an AOI (or the entire screen) to be read from
the disk; a routine to hold a given screen for a specified number of
vertical blanking intervals; and a routine to display the contents of the
image header.
    This module contains the code for the routines.  The header and constant
definitions are in PCVISION.H

*****************************************************************************/



/*****************
 * INCLUDE FILES *
 *****************/

    /*** TURBO C standard include files ***/
#include <conio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <process.h>
#include <string.h>
#include <sys\stat.h>

    /*** module specific include files ***/
#include <constant.h>   /* include required constants, types, etc.  */
#include <toolbox.h>    /* general system routines                   */
#include <pcvision.h>   /* header file for this module               */

FILE=PCVISION.C    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
 55:
 57:
 58: /**********************
 59:  * REQUIREMENTS CHECK *
 60:  **********************/
 61:
 62: #ifndef _BYTE
 63: #error Required type not declared.
 64: #endif
 65: #if !defined(ERROR_COLOR) || !defined(MENU_COLOR)
 66: #error Required constant not declared.
 67: #endif
 68:
 69:
 70:
 71: /************************
 72:  * FUNCTION DEFINITIONS *
 73:  ************************/
 74:
 75: /*-------------------------------------------------------------------------*/
 76: void Clear_screen(byte intensity)
 77:
 78:     /*   This module clears the frame buffer to the given intensity value.  */
 79:
 80:     {
 81:     byte quadrant;                                          /* quadrant to clear        */
 82:
 83:
 84:         /*** select a quadrant; clear one pixel; clear the quadrant;
 85:              and then wait for the clear operation to finish            ***/
 86:     for (quadrant=0; quadrant<4; quadrant++) {
 87:         outportb(BLOCK_SELECT, quadrant);
 88:         pokeb(MEM_BASE, 0x0000, intensity);
 89:         outportb(CON_L, (inportb(CON_L) & 0xCF) | 0x10);
 90:         while(inportb(CON_L) & 0x30);
 91:         }
 92:
 93:     }   /* Clear screen */
 94: /*-------------------------------------------------------------------------*/
 95:
 96:
 97:
 98: /*-------------------------------------------------------------------------*/
 99: void Freeze_mode(void)
100:
101:     /*      This module puts the PCVision out of image digitization mode.   */
102:
103:     {
104:
105:         outportb(CON_L, inportb(CON_L) & 0x0F);
106:
107:     }   /*  Freeze_mode  */
108: /*-------------------------------------------------------------------------*/
109:

FILE=PCVISION.C    Fri Jun 16 01:58:16 1989    PAGE=2
```

```
110:
112:
113: /*-------------------------------------------------------------------------*/
114: void Grab_mode(void)
115:
116:     /*      This module puts the PCVision in image digitization mode.  A
117:         cooresponding call to Freeze_mode will "snap" the picture.          */
118:
119:     {
120:
121:         outportb(CON_L, inportb(CON_L) | 0x30);
122:
123:     }   /*  Grab_mode  */
124: /*-------------------------------------------------------------------------*/
125:
126:
127:
128: /*-------------------------------------------------------------------------*/
129: void Init_board(void)
130:
131:     /*      This module initializes the registers and LUTs on the
132:         PCVision card.                                                      */
133:
134:     {
135:     byte        lut;                /* current LUT number               */
136:     lut_type    lut_array;          /* values to be written in LUT      */
137:     int         val;                /* general index variable           */
138:
139:
140:                 /*** initialize the board's registers ***/
141:     outportb(CON_L, 0x09);
142:     outportb(CON_H, 0x0C);
143:     outportb(LUT_ADDR, 0xFF);
144:     outportb(LUT_DATA, 0xFF);
145:     outportb(MASK, 0x00);           /* enable all planes for writes */
146:     outportb(BLOCK_SELECT, 0x00);   /* select quadrant zero         */
147:
148:                 /*** initialize the LUTs: set each LUT in the first set
149:                      to a linear LUT                                    ***/
150:     for (val=0; val<LUT_SIZE; lut_array[val]=(byte)val++);
151:     for (lut=0; lut<4; lut++) {
152:         Set_lut(lut, GREEN_TABLE);
153:         Write_lut(0, LUT_SIZE-1, lut_array);
154:         Set_lut(lut, INPUT_TABLE);
155:         Write_lut(0, LUT_SIZE-1, lut_array);
156:         }
157:
158:     }   /* Init_board */
159: /*-------------------------------------------------------------------------*/
160:
161:
162:
163: /*-------------------------------------------------------------------------*/
164: void Print_header_info(header_type header)
```

```
165:
167:    /*      This module recieves an image header as an input
168:        parameter.  It then takes the information stored in the
169:        header and prints it in a human readable format using the
170:        conio routines so the color of the text it prints may be
171:        changed by the calling module.                              */
172:
173:    {
174:    byte    comment[257];   /* comment in image */
175:
176:
177:    cprintf("%*cImage width:  %d\r\n", ENTRY_INDENT, ' ', (int)(header[4]+256*header[5]));
178:    cprintf("%*cImage height: %d\r\n", ENTRY_INDENT, ' ', (int)(header[6]+256*header[7]));
179:    cprintf("%*cCoordinates of original X-axis position: (%d,%d)\r\n",
180:            ENTRY_INDENT, ' ', (int)(header[8]), (int)(header[9]));
181:    cprintf("%*cCoordinates of original Y-axis position: (%d,%d)\r\n",
182:            ENTRY_INDENT, ' ',(int)(header[10]), (int)(header[11]));
183:    cprintf("%*cFile type: ", ENTRY_INDENT, ' ');
184:    switch (header[12]+256*header[13]) {
185:        case 0: cprintf("NORMAL (can be correctly read by this program)\r\n");
186:                break;
187:        case 1: cprintf("COMPRESSED (must be uncompressed to be read correctly"
188:                        " by this program)\r\n");
189:                break;
190:        case 2: cprintf("SPECIAL (unspecifed type; must be converted to IMAGEACTION"
191:                        " format to be read correctly by this program\r\n");
192:                break;
193:        default:    cprintf("UNKNOWN (illegal file type)\r\n");
194:    }
195:    memcpy(comment, &header[64], (int)(header[2]+256*header[3]));
196:    comment[header[2]+256*header[3]] = '\0';
197:    cprintf("%*cComment:\r\n", ENTRY_INDENT, ' ');
198:    cprintf("%*c        %s\r\n", ENTRY_INDENT, ' ', comment);
199:
200:    }   /* Print_header_info */
201: /*-------------------------------------------------------------------------*/
202:
203:
204:
205: /*-------------------------------------------------------------------------*/
206: void Read_image(byte quad, char *pathname, header_type header)
207:
208:    /*      This module reads in an AOI (a quadrant of the screen (0-3) or
209:        the entire screen (quad=4).  The header from the image is returned.
210:        The image may be less than a full quadrant but it must be square.
211:        If an image is larger than a quadrant, it is assumed to be a full
212:        screen image.                                                   */
213:
214:    {
215:    quadrant far *buffer;   /* address of screen memory */
216:    int index;              /* general loop variable    */
217:    int handle;             /* handle of input file     */
218:    int val;                /* general loop variable    */
219:
```

```
220:
222:    buffer = MK_FP(MEM_BASE,0);                     /* set address of buffer */
223:
224:    handle = open(pathname, O_RDONLY|O_BINARY);
225:    if (handle==-1) {                               /* does file exist?     */
226:        textcolor(ERROR_COLOR);                     /* no, print error msg  */
227:        cprintf("ERROR: file does not exist (pathname='%s')\r\n", pathname);
228:        textcolor(MENU_COLOR);
229:        }
230:    else {
231:        read(handle, &header[0], 64);               /* read header          */
232:        index = header[2] + 256*header[3];          /* get size of comment  */
233:        read(handle, &header[64], index);           /* read comment         */
234:        val = header[4] + 256*header[5];            /* get X size           */
235:        if (val>256)                                /* must image be loaded */
236:            quad = FULL;                            /* full screen?         */
237:        if (quad!=FULL) {
238:            outportb(BLOCK_SELECT, quad);           /* select quadrant      */
239:            if (val==256) {                         /* if full quadrant     */
240:                read(handle, buffer, 0xFFFE);       /*   read as a big      */
241:                read(handle, buffer-0xFFFE, 0x2);   /*   and a small chunk  */
242:                }
243:            else                                    /* otherwise, read one  */
244:                for (index=0; index<val; index++)   /* line at a time       */
245:                    read(handle, buffer-index*256, val);
246:            }
247:        else {
248:            quad = 0;
249:            for (val=0; val<512; val++) {
250:                outportb(BLOCK_SELECT, quad);                   /* select quadrant  */
251:                read(handle, buffer-(val%256)*256, 0x100);      /* read a quadrant  */
252:                outportb(BLOCK_SELECT, quad+1);                 /* next quadrant    */
253:                read(handle, buffer-(val%256)*256, 0x100);      /* read quadrant    */
254:                if (val>254)
255:                    quad = 2;
256:                }
257:            }
258:        close(handle);
259:        }
260:
261:    }   /* Read_image */
262: /*-------------------------------------------------------------------------*/
263:
264:
265:
266: /*-------------------------------------------------------------------------*/
267: void Save_image(byte quad, char *pathname)
268:
269:    /*      This module saves an AOI (a quadrant of the screen (0-3) or
270:        the entire screen (quad=4).                                     */
271:
272:    {
273:    quadrant far *buffer;   /* address of screen memory */
274:    int handle;             /* handle of input file     */

FILE=PCVISION.C    Fri Jun 16 01:58:16 1989    PAGE=5
```

```
275:     header_type header;     /* image header             */
277:     unsigned long size;     /* size of image            */
278:     int val;                /* general loop variable    */
279:
280:
281:     buffer = MK_FP(MEM_BASE,0);                    /* get address of buffer */
282:
283:
284:     handle = _creat(pathname, FA_ARCH);
285:     if (handle==-1) {                              /* does file exist?     */
286:         textcolor(ERROR_COLOR,);                   /* no  print error msg  */
287:         cprintf("%*cERROR: unable to open file (pathname='%s')", ENTRY_INDENT, ' ', pathname);
288:         textcolor(MENU_COLOR);
289:         print_system_error();
290:         return;
291:         }
292:
293:     if (quad==FULL)
294:         size = 512;
295:     else
296:         size = 256;
297:
298:         /*** create header ***/
299:     header[0] = 'I';
300:     header[1] = 'M';
301:     header[4] = header[6] = (byte) (size % 256);
302:     header[5] = header[7] = (byte) (size / 256);
303:     header[8] = header[9] = header[10] = header[11] = header[12] = header[13] = 0;
304:     header[64] = '\0';
305:     get_input("Enter comm    (255 chars):", "", &header[64]);
306:     if ((val=strlen((char *)&header[64]))>255) {
307:         print_error("Comment too long.  Program corrupted.");
308:         abort();
309:         }
310:     else {
311:         if (val<1) {
312:             header[2] = 1;
313:             header[3] = 0;
314:             }
315:         else {
316:             header[2] = (byte) (val % 256);
317:             header[3] = 0;
318:             }
319:         }
320:     for (val=14; val<64; header[val++] = 0);
321:
322:         /*** write header ***/
323:     write(handle, &header[0], header[2]+64);
324:
325:         /*** write image ***/
326:     if (quad!=FULL) {
327:         outportb(BLOCK_SELECT, quad);       /* select quadrant      */
328:         write(handle, buffer, 0xFFFE);      /*   write as a big     */
329:         write(handle, buffer+0xFFFE, 0x2);  /*   and a small chunk  */
```

```
330:         }
332:     else {
333:         quad = 0;
334:         for (val=0; val<512; val++) {
335:             outportb(BLOCK_SELECT, quad);                    /* select quadrant  */
336:             write(handle, buffer-(val%256)*256, 0x100);      /* write a quadrant */
337:             outportb(BLOCK_SELECT, quad+1);                  /* next quadrant    */
338:             write(handle, buffer-(val%256)*256, 0x100);      /* write quadrant   */
339:             if (val>254)
340:                 quad = 2;
341:             }
342:         }
343:     close(handle);
344:
345:     }   /* Save_image */
346: /*-------------------------------------------------------------------------*/
347:
348:
349:
350: /*-------------------------------------------------------------------------*/
351: void Screen_hold(int num_screens)
352:
353:     /*   This routine waits for the specified number of vertical
354:         blanking periods to occur.  Note: to be effective the call to
355:         this routine must take less than 1.4 ms, also the return must be
356:         less than 1.4 ms  Finally, sync screen changes to the vertical
357:         blanking period by calling this routine with a value of 1.  Note
358:         that a zero screen period is not guaranteed.                    */
359:
360:     {
361:     int index;                                  /* general loop variable       */
362:
363:
364:     for (index=0; index<num_screens; index++) {
365:         while ((inportb(CON_H)&0x04)==0x00);     /* wait until not blanking     */
366:         while ((inportb(CON_H)&0x04)==0x04);     /* wait until blanking starts  */
367:         }
368:
369:     }   /* Screen_hold */
370: /*-------------------------------------------------------------------------*/
371:
372:
373:
374: /*-------------------------------------------------------------------------*/
375: void Set_lut(byte lut, byte lut_set)
376:
377:     /*   This module sets an output LUT, from a LUT set, as the active LUT. */
378:
379:     {
380:
381:     outportb(CON_L, (inportb(CON_L) & 0xF9) | (lut_set<<1));    /* select a LUT set */
382:     if (lut_set==INPUT_TABLE)
383:         outportb(CON_H, (inportb(CON_L) & 0x3F) | (lut<<6));    /* select a LUT     */
384:     else
```

```
385:         outportb(CON_H, (inportb(CON_H) & 0x9F) | (lut<<5));    /* select a LUT     */
387:
388:     }   /* Set_lut */
389: /*-------------------------------------------------------------------------*/
390:
391:
392:
393: /*-------------------------------------------------------------------------*/
394: void Write_lut(byte start, byte stop, lut_type lut)
395:
396:     /*      This module initializes the values from start to stop in a LUT
397:         to the values given in the array.                                  */
398:
399:     {
400:     int val;
401:
402:
403:     for (val=start; val<=stop; val++) {
404:         outportb(LUT_ADDR, val);        /* select LUT entry */
405:         outportb(LUT_DATA, lut[val]);   /* write data       */
406:         }
407:
408:     }   /* Write lut */
409: /*-------------------------------------------------------------------------*/
```

```
/*

            Filename:               pcwrap.h
            Programmer:             Christopher Voltz - UDRI
            Created:                -1/8903.05
            Last Modified:          -1/8904.18
            Interface Protocol:     Turbo C 2.0


    This module was written to act as an interface between the PCVision
routines and the "standard interface" used by the graphics control programs.
The routines simply remap the calls used by the graphics programs.  Any
parameters which are not relevant to the PCVision hardware are simply ignored.
The PCVision provides two "buffers" by using different LUT's.  The NOISE_BUFFER
is actually one LUT while the SIGNAL_BUFFER is an actual buffer with its own
LUT.

*/


#include <pcvision.h>


#define OUT_LUT_SIZE LUT_SIZE


byte clear_screen(byte intensity, byte screen);
void display(byte state);
void display_buffer(byte buffer);
void freeze_mode(void);
void grab_mode(void);
byte init_board(void);
void print_header_info(header_type header);
byte read_image(byte buffer, byte quad, char *pathname, header_type header);
byte read_2_images(byte buffer, char *pathname_1, header_type header_1,
                   byte quad,   char *pathname_2, header_type header_2);
void save_image(byte buffer, byte quad, char *pathname);
void screen_hold(word num_fields);
void set_write_protect(byte value);
void write_lut(byte type_of_lut, byte lut_number, word start, word stop,
               lut_type *lut);
```

```
/*

            Filename:               pcwrap.c
            Programmer:             Christopher Voltz - UDRI
            Created:                -1/8903.05
            Last Modified:          -1/8904.18
            Interface Protocol:     Turbo C 2.0


    This module was written to act as an interface between the PCVision
routines and the "standard interface" used by the graphics control programs.
The routines simply remap the calls used by the graphics programs.  Any
parameters which are not relevant to the PCVision hardware are simply ignored.
The PCVision provides two "buffers" by using different LUT's.  The NOISE_BUFFER
is actually one LUT while the SIGNAL_BUFFER is an actual buffer with its own
LUT.

*/



#include <constant.h>
#include <pcwrap.h>

#pragma warn -par

byte clear_screen (byte intensity, byte screen)
{
    int         index;
    lut_type    lut_array;


    if (screen==NOISE_BUFFER)
    {
        for (index=0; index<LUT_SIZE; lut_array[index++]=intensity)
            ;
        Set_lut(NOISE_BUFFER, GREEN_TABLE);
        Write_lut(0, LUT_SIZE-1, lut_array);
    }
    Clear_screen(intensity);
    return 0;
}   /* clear_screen */


void display (byte state)
{
    /* nothing */
}   /* display */


void display_buffer (byte buffer)
{
    if (buffer==SIGNAL_BUFFER)
```

```
55:         Set_lut(SIGNAL_BUFFER, GREEN_TABLE);
57:     else
58:         Set_lut(NOISE_BUFFER, GREEN_TABLE);
59: }   /* display_buffer */
60:
61:
62: void freeze_mode (void)
63: {
64:     Freeze_mode();
65: }   /* freeze_mode */
66:
67:
68: void grab_mode (void)
69: {
70:     Grab_mode();
71: }   /* grab_mode */
72:
73:
74: byte init_board (void)
75: {
76:     Init_board();
77:     return 0;
78: }   /* init_board */
79:
80:
81: void print_header_info (header_type header)
82: {
83:     Print_header_info (header);
84: }   /* print_header_info */
85:
86:
87: byte read_image (byte buffer, byte quad, char *pathname, header_type header)
88: {
89:     if (buffer==SIGNAL_BUFFER)
90:     {
91:         Read_image(quad, pathname, header);
92:         return 0;
93:     }
94:     else
95:         return 1;
96: }   /* read_image */
97:
98:
99: byte read_2_images (byte buffer, char *pathname_1, header_type header_1,
100:                     byte quad,   char *pathname_2, header_type header_2)
101: {
102:     if (buffer==SIGNAL_BUFFER)
103:     {
104:         Read_image(0, pathname_1, header_1);
105:         Read_image(1, pathname_2, header_2);
106:         return 0;
107:     }
108:     else
109:         return 1;
```

```
110: }   /* read_2_images */
112:
113:
114: void save_image (byte buffer, byte quad, char *pathname)
115: {
116:     Save_image(quad, pathname);
117: }   /* save_image */
118:
119:
120: void screen_hold (word num_fields)
121: {
122:     Screen_hold (num_fields>>1);
123: }   /* screen_hold */
124:
125:
126: void set_write_protect (byte value)
127: {
128:     /* nothing */
129: }   /* set_write_protect */
130:
131:
132: void write_lut (byte type_of_lut, byte lut_number, word start, word stop,
133:                 lut_type *lut)
134: {
135:     Write_lut (start, stop, *lut);
136: }   /* write_lut */
137:
138:
139: #pragma warn .par
```

```
/*****************************************************************************

        FILENAME:               response.h
        CREATED:                -1/8805.23
        LAST MODIFIED:          -1/8810.27
        PROGRAMMER:             Christopher Voltz - UDRI
        INTERFACE PROTOCOL:     TURBO C 2.0
        USAGE:              program.c:
                                #include <response.h>

                            constant.h:
                                #define ERROR_COLOR LIGHTRED + BLINK
                                #define MENU_COLOR WHITE
                                #define OPTION_COLOR CYAN
                                #define R_BIT_MASK 0x88 [0xF0]
                                #define R_BUTTON_1 0x80
                                #define R_BUTTON_2 0x08 [0x40]
                                [#define R_FOUR_BUTTONS 1]
                                [#define R_BUTTON_3 0x20 ]
                                [#define R_BUTTON_4 0x10 ]
                                #define R_PORT     0x0379
                                #define _BYTE 1
                                typedef unsigned char byte
                                #endif

                                [optional]


    This module allows the user to read the two buttons connected through the
response box to the port designated by the calling program.  The included
routines are:

1) get_response => this routine reads data from the port, ANDs it against the
   given bitmask, goes into a loop where it continously reads data from the
   port and ANDs it to the bitmask until the new data is different from the
   old data (ie. a change in states is detected), XORs the new data with the
   old byte of data to set the bits which have changed and returns that byte
   of data.   Note: if any key is pressed on the keyboard, this routine will
   terminate immediately without removing the keystroke from the keyboard
   buffer.
2) test_response_box => this routine reads a response from the response box,
   compares it to the two legal values specified by the main program, and
   prints a message indicating which button was pressed or an error message if
   the bit pattern was unrecognized.


    The calling program must initialize the following constants required:
R_BIT_MASK  => the bit pattern to AND the input data with; used to clear extra
               bits so only relevant bits are used, eg.:
               #define R_BIT_MASK 0x88  * for a response box which responds *
                                        * with bits 7 and 3 set             *
R_BUTTON_1  => the bit pattern which represents the first response button being
               pressed, eg.:
               #define R_BUTTON_1 0x80  * if bit 7 is used for button 1  *

FILE=RESPONSE.H    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
R_BUTTON_2  => the bit pattern which represents the second response button
               being pressed, eg.:
               #define R_BUTTON_2 0x08  * if bit 3 is used for button 2  *
R_PORT      => the address of the port to read the data from, eg.:
               #define R_PORT 0x0379    * parallel port one (LPT1:)      *


    Note that due to the method used to read the data, any type of switch
and/or port combination may be used if the pins representing the status of the
buttons are constantly driven.  That is, a serial or a parallel port may be
used with equal ease if the correct data port addresses are given.  Also, if
the button is normally open, normally closed, or momentary, it will interface
with these routines as they detect the change in states which must result from
a button open or closure; however, if the button is momentary, the routines
could respond to two results if the user opened and closed (or vice versa) the
switches slow enough that the main program had time to call these routines
again.
eg.: two momentary switches shall be connected to a parallel port, a sample
     circut might be, (assume the response box does not drive the lines and
     that the switches are mechanically debounced)

              pin 15                  pin 11
                |                       |
pin 1 ==========+                       +========== pin 16
                |                       |
    switch 1 -||          switch 2 -||  |
                |                       |
                \                       \
                /                       /
                \  R = 1 kohm           \   R = 1 kohm
                /                       /
                |                       |
                +=======================+========== pin 19

    recall: pin 1   =>  -STROBE (provides +5V (high) for switch 1)
            pin 11  => BUSY     (input for switch 1 => bit 7 of status word)
            pin 15  => -ERROR   (input for switch 2 => bit 3 of status word)
            pin 16  => -INIT    (provides +5V (high) for switch 2)
            pin 19  => GROUND   (provides ground (low) for switches)

    If we wished to connect this to LPT1: and we wished switch 1 to represent
button 1 and switch 2 to represent button 2, then we would define the system
parameters as shown in the previous examples for setting them.

eg.: four SPDT switches shall be connected to a parallel port, a sample circut
     might be, (assume the response box drives the lines and the switches must
     be electronically debounced)

                              -  Vcc (+5V)
                              |
                              \
                              /
                              \  R = 10 kohm
                              /
```

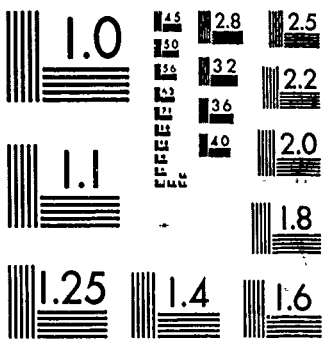
```
110:                                    |
112:                  O==========+======+
113:                 /                        (2 input NAND) +======= (not connected)
114:                /                      +======\      /======
115:      +========O  switch 1                      \===/==\
116:      |                                 +=========/     \========= pin 11
117:      |                                   (2 input NAND) +
118:      |               O==========+======+
119:      V GND                      |
120:                                 \
121:                                 /
122:                                 \  R = 10 kohm
123:                                 /
124:                                 |
125:                                 V  Vcc (+5V)
126:     ------------------------------------------------------------------------
127:     repeat above circut for each switch connecting further ouputs to pins 10,
128:     12, and 13.  (one additional switch could be connected to pin 11)  The
129:     debouncer's ground should also be connected to the input signal ground, pin
130:     11.
131:     ------------------------------------------------------------------------
132:
133:     Thus, we have debounced the switch using an R-S bistable circut.  The two
134:     NANDs can be obtained using only one 7400 quad 2 input NAND gate chip.
135:     Therefore, the above circut requires two 7400s.  A suitable power supply
136:     must also be constructed but this is trivial.  An 8544 would have been
137:     more suitable for the four switches arrangement but the 7400s were in
138:     house stock.
139:         We must configure the software by defining the following:
140:     #define R_FOUR_BUTTONS  1       * notify software 4 buttons in use      *
141:     #define R_BIT_MASK  0xF0        * the response box will set bits 7-4    *
142:     #define R_BUTTON_1  0x80        * switch 1 is connected to bit 7        *
143:     #define R_BUTTON_2  0x40        * switch 2 is connected to bit 6        *
144:     #define R_BUTTON_3  0x20        * switch 3 is connected to bit 5        *
145:     #define R_BUTTON_4  0x10        * switch 4 is connected to bit 4        *
146:    (#define R_BUTTON_5  0x08        * switch 5 is connected to bit 5 if it
147:                                      is connected at all                    *)
148:     #define R_PORT      0x0379      * response box is connected to LPT1:    *
149:
150:         This module contains only the constants, type definitions, and
151:     function prototypes for the given routines.  The code for these routines
152:     is contained in the file RESPONSE.C
153:
154: ****************************************************************************/
155:
156:
157: /*******************
158:  * TYPE DEFINTIONS *
159:  *******************/
160:
161: #ifndef _BYTE                   /* define a byte type */
162: #define _BYTE 1
163: typedef unsigned char byte
164: #endif
```

```
165:
167:
168: /************************
169:  * FUNCTION PROTOTYPES *
170:  ************************/
171:
172: byte get_response(void);
173: void test_response_box(void);
```

```
/*****************************************************************************

        FILENAME:               response.c
        CREATED:                -1/8803.03
        LAST MODIFIED:          -1/8810.27
        PROGRAMMER:             Christopher Voltz - UDRI
        INTERFACE PROTOCOL:     TURBO C 2.0
        USAGE:                program.c:
                                    #include <response.h>

                              constant.h:
                                    #define ERROR_COLOR LIGHTRED + BLINK
                                    #define MENU_COLOR WHITE
                                    #define OPTION_COLOR CYAN
                                    #define R_BIT_MASK 0x88 [0xF0]
                                    #define R_BUTTON_1 0x80
                                    #define R_BUTTON_2 0x08 [0x40]
                                    [#define R_FOUR_BUTTONS 1]
                                    [#define R_BUTTON_3 0x20 ]
                                    [#define R_BUTTON_4 0x10 ]
                                    #define R_PORT     0x0379
                                    #ifndef _BYTE
                                    #define _BYTE 1
                                    typedef unsigned char byte
                                    #endif

                                    [optional]


    This module allows the user to read the two buttons connected through the
response box to the port designated by the calling program.  The included
routines are:

1) get_response => this routine reads data from the port, ANDs it against the
   given bitmask, goes into a loop where it continously reads data from the
   port and ANDs it to the bitmask until the new data is different from the
   old data (ie. a change in states is detected), XORs the new data with the
   old byte of data to set the bits which have changed and returns that byte
   of data.   Note: if any key is pressed on the keyboard, this routine will
   terminate immediately without removing the keystroke from the keyboard
   buffer.
2) test_response_box => this routine reads a response from the response box,
   compares it to the two legal values specified by the main program, and
   prints a message indicating which button was pressed or an error message if
   the bit pattern was unrecognized.


    The calling program must initialize the following constants required:
R_BIT_MASK  => the bit pattern to AND the input data with; used to clear extra
               bits so only relevant bits are used, eg.:
               #define R_BIT_MASK 0x88  * for a response box which responds *
                                        * with bits 7 and 3 set             *
R_BUTTON_1  => the bit pattern which represents the first response button being
               pressed, eg.:

FILE=RESPONSE.C    Fri Jun 16 01:58:16 1989    PAGE=1
```

```
                #define R_BUTTON_1 0x80  * if bit 7 is used for button 1  *
R_BUTTON_2  => the bit pattern which represents the second response button
               being pressed, eg.:
               #define R_BUTTON_2 0x08  * if bit 3 is used for button 2  *
R_PORT      => the address of the port to read the data from, eg.:
               #define R_PORT 0x0379    * parallel port one (LPT1:)      *


    Note that due to the method used to read the data, any type of switch
and/or port combination may be used if the pins representing the status of the
buttons are constantly driven.  That is, a serial or a parallel port may be
used with equal ease if the correct data port addresses are given.  Also, if
the button is normally open, normally closed, or momentary, it will interface
with these routines as they detect the change in states which must result from
a button open or closure; however, if the button is momentary, the routines
could respond to two results if the user opened and closed (or vice versa) the
switches slow enough that the main program had time to call these routines
again.
eg.: two momentary switches shall be connected to a parallel port, a sample
     circut might be, (assume the response box does not drive the lines and
     that the switches are mechanically debounced)

              pin 15                  pin 11
                |                       |
pin 1 ==========+                       +========== pin 16
                |                       |
    switch 1 -||          switch 2 -||  |
                |                       |
                \                       \
                /                       /
                \  R = 1 kohm           \   R = 1 kohm
                /                       /
                |                       |
                +=======================+========== pin 19

    recall: pin 1   =>  -STROBE (provides +5V (high) for switch 1)
            pin 11  => BUSY     (input for switch 1 => bit 7 of status word)
            pin 15  => -ERROR   (input for switch 2 => bit 3 of status word)
            pin 16  => -INIT    (provides +5V (high) for switch 2)
            pin 19  => GROUND   (provides ground (low) for switches)

    If we wished to connect this to LPT1: and we wished switch 1 to represent
button 1 and switch 2 to represent button 2, then we would define the system
parameters as shown in the previous examples for setting them.

eg.: four SPDT switches shall be connected to a parallel port, a sample circut
     might be, (assume the response box drives the lines and the switches must
     be electronically debounced)

                            ^  Vcc (+5V)
                            |
                            \
                            /
                            \  R = 10 kohm
```

```
110:                          /
112:                          |
113:                 O========+=====+
114:                /                  (2 input NAND) +======= (not connected)
115:               /                +=====\      /=====
116:      +=======O  switch 1             \===/==\
117:      |                         +=========/    \========= pin 11
118:      |                         (2 input NAND) +
119:      |          O==========+=====+
120:      V GND                |
121:                           \
122:                           /
123:                           \  R = 10 kohm
124:                           /
125:                           |
126:                           V  Vcc (+5V)
127:     ------------------------------------------------------------------------
128:     repeat above circut for each switch connecting further ouputs to pins 10,
129:     12, and 13.  (one additional switch could be connected to pin 15)  The
130:     debouncer's ground should also be connected to the input signal ground, pin
131:     19.
132:     ------------------------------------------------------------------------
133:
134:     Thus, we have debounced the switch using an R-S bistable circut.  The two
135:     NANDs can be obtained using only one 7400 quad 2 input NAND gate chip.
136:     Therefore, the above circut requires two 7400s.  A suitable power supply
137:     must also be constructed but this is trivial.  An 8544 would have been
138:     more suitable for the four switches arrangement but the 7400s were in
139:     house stock.
140:         We must configure the software by defining the following:
141:     #define R_FOUR_BUTTONS  1       * notify software 4 buttons in use      *
142:     #define R_BIT_MASK  0xF0        * the response box will set bits 7-4    *
143:     #define R_BUTTON_1  0x80        * switch 1 is connected to bit 7        *
144:     #define R_BUTTON_2  0x40        * switch 2 is connected to bit 6        *
145:     #define R_BUTTON_3  0x20        * switch 3 is connected to bit 5        *
146:     #define R_BUTTON_4  0x10        * switch 4 is connected to bit 4        *
147:    (#define R_BUTTON_5  0x08        * switch 5 is connected to bit 5 if it
148:                                      is connected at all                    *)
149:     #define R_PORT      0x0379      * response box is connected to LPT1:    *
150:
151:         This module the code for these routines.  The constants, type
152:     definitions, and function prototypes for the given routines are in the
153:     the file RESPONSE.H
154:
155: ****************************************************************************/
156:
157:
158:
159: /*****************
160:  * INCLUDE FILES *
161:  *****************/
162:
163:     /*** standard TURBO C include files ***/
164: #include <conio.h>

FILE=RESPONSE.C    Fri Jun 16 01:58:16 1989    PAGE=3
```



```
220:              to be pressed;  if a switch changed state, get final state ***/
222:     do
223:         state_2 = inportb(R_PORT) & R_BIT_MASK;
224:     while (state_1==state_2 && !kbhit());
225:
226:         /*** return change in states ***/
227:     return(state_1 ^ state_2);
228:
229:     }   /* get_response */
230: /*-------------------------------------------------------------------------*/
231:
232:
233:
234: /*-------------------------------------------------------------------------*/
235: void test_response_box(void)
236:
237:     /*      This module tests the response box to make sure it is
238:         functioning correctly.                                          */
239:
240:     {
241:     byte response;      /* subject's response */
242:
243:
244:         /*** setup screen ***/
245:     clrscr();
246:     textcolor(MENU_COLOR);
247:     cprintf("TESTING SWITCHBOX: \n\r\n\r");
248:     cprintf("Press any keyboard key to end.\n\r");
249:     cprintf("Press any switchbox key to see response.\n\r\n\rBEGIN:\n\r");
250:
251:         /*** get response and print it or an error message, repeat
252:              until the user presses a keyboard key                      ***/
253:     textcolor(OPTION_COLOR);
254:     while (!kbhit()) {
255:         response = get_response();
256:         if (response == R_BUTTON_1)
257:             cprintf("\t Button 1.\n\r");
258:         else if (response == R_BUTTON_2)
259:             cprintf("\t Button 2.\n\r");
260: #if defined(R_FOUR_BUTTONS)
261:         else if (response == R_BUTTON_3)
262:             cprintf("\t Button 3.\n\r");
263:         else if (response == R_BUTTON_4)
264:             cprintf("\t Button 4.\n\r");
265: #endif
266:         else if (!kbhit()) {
267:              textcolor(ERROR_COLOR & !BLINK);
268:              cprintf("\t ERROR: (switch pattern=%2xX)\n\r", (int)response);
269:              textcolor(OPTION_COLOR);
270:              }
271:         }   /* while */
272:     getch();
273:
274:     }   /* test_response_box */
```

```
275: /*-------------------------------------------------------------------------*/
```

```
 1: /*****************************************************************************
 2:
 3:             FILENAME:               toolbox.h
 4:             PROGRAMMER:             Christopher Voltz - UDRI
 5:             CREATED:                -1/8805.23
 6:             LAST MODIFIED:          -1/8904.14
 7:             INTERFACE PROTOCOL:     TURBO C 2.0
 8:             USAGE:                program.c:
 9:                                       #include <toolbox.h>
10:
11:                                   constant.h:
12:                                       #define ENTRY_INDENT    5
13:                                       #define ENTRY_COLOR     YELLOW
14:                                       #define ERROR_COLOR     LIGHTRED + BLINK
15:                                       #define ESC_KEY         27
16:                                       #define MENU_COLOR      WHITE
17:                                       #define OPTION_COLOR    CYAN
18:
19:
20:         This module provides routines to confirm an entry; print an error
21:     message and wait for confirmation; to print a menu option, with the
22:     activating key highlighted; to swap elements; to swap two integers; to
23:     print screen titles centered and in the menu color; and to input values.
24:         This module contains only the constant definitions and function
25:     prototypes.  The actual code for the routines is in TOOLBOX.C
26:
27: *****************************************************************************/
28:
29:
30:
31: /***********************
32:  * FUNCTION PROTOTYPES *
33:  ***********************/
34:
35: char confirm(void);
36: void get_input(char *prompt, char *format, ...);
37: void print_error(char *err_msg);
38: void print_option(char *option);
39: void print_system_error(void);
40: void print_title(char *text);
41: void swap(void *n1, void *n2, unsigned int len);
42: void swap_int(int *param_1, int *param_2);
```

```
/*****************************************************************************

        FILENAME:               toolbox.c
        PROGRAMMER:             Christopher Voltz - UDRI
        CREATED:                -1/8803.08
        LAST MODIFIED:          -1/8904.14
        INTERFACE PROTOCOL:     TURBO C 2.0
        USAGE:              program.c:
                                #include <toolbox.h>

                            constant.h:
                                #define ENTRY_INDENT    5
                                #define ENTRY_COLOR     YELLOW
                                #define ERROR_COLOR     LIGHTRED + BLINK
                                #define ESC_KEY         27
                                #define MENU_COLOR      WHITE
                                #define OPTION_COLOR    CYAN


            This module provides routines to confirm an entry; print an error
        message and wait for confirmation; to print a menu option, with the
        activating key highlighted; to swap two integers; to print screen
        titles centered and in the menu color; and to input values.
            This module contains the code for the routines.  The constants and
        type definitions are contained in TOOLBOX.H

***************** ***************************************************************/


/*****************
 * INCLUDE FILES *
 *****************/

    /*** standard TURBO C include files ***/
#include <conio.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

    /*** module specific include files ***/
#include <constant.h>   /* system specific constants    */
#include <toolbox.h>    /* header file for this module  */


/**********************
 * REQUIREMENTS CHECK *
 **********************/

#if !defined(ENTRY_INDENT) || !defined(ERROR_COLOR) || !defined(ESC_KEY)
#error Required constant not defined.
#endif
```

```
#if !defined(MENU_COLOR) || !defined(OPTION_COLOR) || !defined(ENTRY_COLOR)
#error Required constant not defined.
#endif


/************************
 * FUNCTION DECLARATIONS *
 ************************/

/*---------------------------------------------------------------------------*/
char confirm(void)

    /*      This module prints the confirm message and waits for the
        user to press a key.  The key is returned.                           */

    {
    int  col;           /* column cursor is on when called  */
    char response;      /* character to return              */
    int  row;           /* row cursor is on when called     */


        /*** save cursor position ***/
    col = wherex();
    row = wherey();

        /*** display error message ***/
    textcolor(ERROR_COLOR);
    cprintf("\n\r\n%*c\aCONFIRM <Y/N>\n\r", ENTRY_INDENT, ' ');
    textcolor(MENU_COLOR);
    response = getch();

        /*** clear error message and return cursor to original position */
    gotoxy(col, row);
    cprintf("%*c", 80*(wherey()-row+1), ' ');
    gotoxy(col, row);

    return(response);

    }   /* confirm */
/*---------------------------------------------------------------------------*/


/*---------------------------------------------------------------------------*/
void get_input(char *prompt, char *format, ...)

    /*      This module prints the given prompt message and reads in
        a given set of variables from the console using the given
        format string.  If the format string is NULL then the input is
        returned exactly as entered but without the trailing CR -- use
        for inputting strings where data may be separated by spaces;
        returns one string.                                              */

```

```
110:     {
112:     va_list argptr;
113:     char    ch;
114:     int     index;
115:     char    input[257];
116:
117:
118:     textcolor(MENU_COLOR);
119:     cprintf("%*c%s ", ENTRY_INDENT, ' ', prompt);
120:     textcolor(ENTRY_COLOR);
121:     input[0] = '\0';
122:     for (index=0; (ch=getch())!='\x0D';)
123:         switch (ch) {
124:             case '\b':      if (index!=0) {
125:             /* backspace */     input[--index] = '\0';
126:                                 cprintf("\b \b");
127:                                 }
128:                             break;
129:             case '\x1B':    putch('\\');
130:             /* ESCape */        if (wherey()==25)
131:                                 cprintf("\n\r");
132:                             gotoxy(wherex()-strlen(input)-1, wherey()+1);
133:                             input[index=0] = '\0';
134:                             break;
135:             case '\x03':    putch('\n');
136:             /* ctrl-C */    abort();
137:             default:        if (index<256) {
138:                                 putch(ch);
139:                                 input[index++] = ch;
140:                                 input[index] = '\0';
141:                                 }
142:                             else putch('\a');
143:                             break;
144:             }
145:     textcolor(MENU_COLOR);
146:     cprintf("\n\r");
147:     input[index++] = ch;
148:     input[index] = '\0';
149:
150:     va_start(argptr, format);
151:     if (strlen(format)==0) {
152:         input[index-1] = '\0';                  /* remove trailing CR */
153:         memccpy(va_arg(argptr, char *), input, 0, strlen(input)+1);
154:         }
155:     else
156:         vsscanf(input, format, argptr);
157:     va_end(argptr);
158:
159:     }   /* get_input */
160: /*---------------------------------------------------------------------------*/
161:
162:
163:
164: /*---------------------------------------------------------------------------*/
```

```
165: void print_error(char *err_msg)
167:
168:     /*      This module prints the given error message, waits for the
169:         user to press the ESC key, and then returns.                    */
170.
171:     {
172:     int col;        /* column cursor is on when called  */
173:     int row;        /* row cursor is on when called     */
174:     int val;        /* number of rows to clear          */
175:
176:
177:         /*** save cursor position ***/
178:     col = wherex();
179:     row = wherey();
180:
181:         /*** display error message ***/
182:     textcolor(ERROR_COLOR);
183:     cprintf("\n\r%*c%s\n\r", ENTRY_INDENT, ' ', err_msg);
184:
185:         /*** get user confirmation ***/
186:     textcolor(MENU_COLOR);
187:     cprintf("%*cPress <ESC> to continue", ENTRY_INDENT, ' ');
188:     while (kbhit()) getch();                                    /* clear buffer */
189:     while (getch()!=ESC_KEY);
190:
191:         /*** clear error message and return cursor to original position */
192:     val = wherey()-row+1;
193:     gotoxy(col, row);
194:     textcolor(ENTRY_COLOR);
195:     cprintf("%*c", 80*val, ' ');
196:     gotoxy(col, row);
197:     textcolor(MENU_COLOR);
198:
199:     }   /* print_error */
200: /*-------------------------------------------------------------------------*/
201:
202:
203:
204: /*-------------------------------------------------------------------------*/
205: void print_option(char *option)
206:
207:     /*      This module prints the given string highlighting the
208:         specified option key.  The string to be highlighted should be
209:         at the beginning of the input string and separated from the rest
210:         of the string by a pipe "|".                                    */
211:
212:     {
213:     int     index;
214:     char    high_text[255];
215:     char    *norm_text;
216:
217:
218:         /*** get text to be highlighted ***/
219:     norm_text = strdup(option);

FILE=TOOLBOX.C    Fri Jun 16 01:58:16 1989    PAGE=4
```

```
220:     for (index=0; norm_text[index]!='|' && index<254 && norm_text[index]!='\0'; index++)
222:         high_text[index] = norm_text[index];
223:     high_text[index++] = '\0';
224:
225:         /*** remove text to highlight from original string ***/
226:     norm_text = strdup(norm_text+index);
227:
228:         /*** find where highlighted text begins ***/
229:     for (index=0; index<254 && norm_text[index]!='\0' &&
230:          strncmp(high_text, norm_text+index, strlen(high_text))!=0; index++);
231:
232:         /*** print left portion of text and remove it from string ***/
233:     textcolor(MENU_COLOR);
234:     cprintf("%*c", ENTRY_INDENT, ' ');
235:     if (index!=0){
236:         cprintf("%.*s", index  norm_text);
237:         norm_text = strdup(norm_text+index);
238:         }
239:
240:         /*** print highlighted text ***/
241:     textcolor(OPTION_COLOR);
242:     cprintf("%s", high_text);
243:
244:         /*** print right part of text ***/
245:     textcolor(MENU_COLOR);
246:     cprintf("%s\n\r", norm_text+strlen(high_text));
247:
248:     }   /* print_option */
249: /*---------------------------------------------------------------------------*/
250:
251:
252:
253: /*---------------------------------------------------------------------------*/
254: void print_system_error(void)
255:
256:     /*      This routine determines the system error message, creates
257:         an error message string, and calls the print error routine to
258:         display the message and confirm that the user has seen it.      */
259:
260:     {
261:
262:     print_error(sys_errlist[errno]);
263:
264:     }   /* print_system_error */
265: /*---------------------------------------------------------------------------*/
266:
267:
268:
269: /*---------------------------------------------------------------------------*/
270: void print_title(char *text)
271:
272:     /*      This module prints the given text, in the menu color,
273:         centered at the top of the screen, which it clears.         */
274:
```

```
275:     {
277:
278:     clrscr();
279:     textcolor(MENU_COLOR);
280:     cprintf("%*c%s\n\r", (80 - strlen(text)) / 2, ' ', text);
281:
282:     }   /* print_title */
283: /*-------------------------------------------------------------------------*/
284:
285:
286:
287: /*-------------------------------------------------------------------------*/
288: void swap (void *n1, void *n2, unsigned int len)
289:
290:     /*      This module recieves two pointers to objects which need to be
291:         swapped as well as the number of bytes to swap.  If there was not
292:         enough memory to swap the elements, nothing happens.               */
293:
294:     {
295:     void *temp;     /* temporary pointer */
296:
297:     temp = malloc(len);
298:     if (!temp)
299:         return;
300:     memcpy(temp, n1, len);
301:     memcpy(n1, n2, len);
302:     memcpy(n2, temp, len);
303:     free (temp);
304:
305:     }   /* swap */
306: /*-------------------------------------------------------------------------*/
307:
308:
309:
310: /*-------------------------------------------------------------------------*/
311: void swap_int(int *param_1, int *param_2)
312:
313:     /*      This module recieves pointers to two integers.  It swaps
314:         the two integers so param_1 will be equal to what param_2
315:         originally was and vice versa.                                     */
316:
317:     {
318:     int temp;           /* temporary storage    */
319:
320:     temp = *param_1;
321:     *param_1 = *param_2;
322:     *param_2 = temp;
323:
324:     }   /* swap_int */
325: /*-------------------------------------------------------------------------*/
```


* U. S. GOVERNMENT PRINTING OFFICE: 1990--761-051/20003